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

34. konference

Sborník příspěvků

Sport Hotel Kurzovní
Praděd
17.–20. května 2009

Programový výbor

Vladimír Rudolf, Západočeská univerzita v Plzni
Jakub Urbanec, HP Praha
Václav Pergl, Kerio Technologies

Sborník příspěvků z 34. konference EurOpen.CZ, 17.–20. května 2009

Plzeň 2009. 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 1 160/14, Plzeň

Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-16-7

Obsah

Xen

Michal Švamberg

E-mail: svamberg@civ.zcu.cz

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

Právě možnost provozu virtuálních strojů v produkčním prostředí způsobila velkou oblibu virtualizace v posledním období. Nejčastěji zmiňované projekty v tomto světle jsou Xen, Linux VServer a KVM. O tom proč jsme na ZČU vybrali pro provoz virtuálních strojů právě Xen, jakou mají konfiguraci a s jakými zkušenostmi je provozujeme se dozvíte právě v tomto příspěvku.

Virtualizace

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

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

Nevýhodou tohoto pojetí virtualizace je vysoká režie vzniklá vlivem softwarového překladu instrukcí nebo I/O operací. Existuje několik optimalizací (dynamická rekompilace, přímé vykonání instrukce), avšak režie je i tak vysoká.

Typickými zástupci jsou BOSCH, QEMU nebo DOSEMU.

Virtualizace na úrovni OS kde je virtuálnímu stroji předložena část fyzického stroje. Jejich vzájemná izolace je zajištěna vytvořením bariér mezi těmito systémy. Všechny virtuální stroje běží nad společným jádrem, nelze tedy mít virtuální stroje různých operačních systémů. Typickým zástupcem tohoto způsobu virtualizace je VServer.

Paravirtualizace je cesta, jak spojit to lepší z obou předchozích přístupů. Virtuální stroj se nesnaží hardware plně emulovat, namísto toho poskytuje hostovanému systému speciální rozhraní, které k němu umožní přistupovat.

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

Paravirtualizace není jen doménou Xenu, tuhle myšlenku používá také User Mod Linux (UML). Ovšem UML bylo konstruováno za účelem ladění a vývoje linuxového jádra, proto není optimalizováno na výkon a do produkčního prostředí se nehodí.

Dále existuje skupina *aplikační virtualizace* do níž patří například Wine. Známý VMWare, ale nejen on, spadá do skupiny *nativní virtualizace*, což je plná virtualizace s částečnou emulací hardware. Aby byla zachována únosná režie je třeba, aby architektura hostitele byla totožná s virtuálním strojem. V případě této virtualizace není potřeba provádět modifikace hostovaného systému.

Xen

Umožňuje bezpečný provoz několika virtuálních strojů – každý se svým vlastním operačním systémem – na jediném fyzickém stroji. Výkon virtuálních strojů spravovaných Xenem se blíží jejich výkonu při nativním chodu.

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

Ve své podstatě je Xen sada patchů pro jádra Linuxu[1] či OpenBSD a několik skriptů, kterými lze virtualizaci ovládat. Základním kamenem je *hypervisor*. Hypervisor je malá část jádra, která má za úkol řídit nedělitelné operace. Je to zvláště plánování procesorového času, řízení přerušení, mapování stránek či bootovací sekvence stroje a start *Domain-0*, tzv. privilegovaného stroje.

Pro řízení Xenu je třeba mít odpovídající prostředí, nejlépe s plnohodnotným operačním systémem a ovladači k hardwaru. Tuto službu nám zajišťuje *Domain-0*, který se startuje po zavedení hypervisora. Ovladače se v pojetí Xenu dělí na dvě skupiny: *front-end* a *back-end*. Ovladače typu *back-end* spravují přístup

[1] Do verze 2.6.23 byla přidána podpora domU (neprivilegovaná doména). Od verze 2.6.30 by měla být zařazena kompletní podpora včetně dom0 (privilegovaná doména).

k fyzickému zařízení, ke každému back-end ovladači tedy přísluší nějaký běžný ovladač. *Front-end* ovladač je pak jakási trubka, která se z jiných strojů připojuje k back-end ovladači spravující normální ovladač. Nejběžnější způsob je, že back-end ovladače jsou v Domain-0 a front-end ovladače ve virtuálních strojích. Vzhledem k tomu, že události v systému nám spravuje hypervisor, je možné, aby zařízení bylo „exportováno“ do samostatného virtuálního stroje, který bude přes back-end ovladač zpřístupňovat toto zařízení dalším virtuálním strojům.

Aby to bylo ještě zajímavější, lze mezi sebou kombinovat různé druhy systémů. Například na Domain-0 s jádrem 2.6 lze provozovat linuxové stroje s jádry 2.4 i 2.6 a navíc si jako bonus pustit OpenBSD. S nástupem Xenu 3.0 se do toho přimíchává 64bitová architektura a také možnost spouštět nemodifikované systémy. Tím se otevírá další možnost využití Xenu – vývojář softwaru již nebude potřebovat pro testování své aplikace více strojů, ale bude mu stačit jeden s mnoha virtuálními stroji.

Xen je založen na principu paravirtualizace. S nízkou režií je rozumně univerzální. Vzhledem k nutnosti zásahu do zdrojových kódů jej lze použít jen s operačními systémy, které jsou otevřené. Ve verzi Xen 3.0 přibyla podpora hardwarové virtualizace, s jejíž kombinací je možné provozovat též neupravené hostované systémy. Technologie IntelVT obsahuje podporu virtualizace v samotném hardware. Zjednodušeně lze říci, že tyto hardwarové čipy maskují existenci hypervisora jako správce prostředků, čímž si hostovaný operační systém myslí, že má celý systém pod kontrolou.

Zde se jedná o tzv. *kruhy ochrany (rings)*, které jsou číslovány od 0 ke 3. Operační systémy jsou zvyklé běžet v kruhu 0 a rozhodovat o prostředcích počítače. Ale při virtualizaci už je v kruhu 0 zaveden hypervisor a ten spouští hosty v kruhu 1. Zamaskovat tento problém pro nemodifikované operační systémy lze pouze při spolupráci softwarových a hardwarových prostředků. V kruhu 3 jsou spouštěny aplikace, kruh 2 nebývá využíván.

Přínosy Xenu

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

Dalším přínosem je jednodušší správa. Snadno vytvoříte nový stroj, nemusíte napřed shánět železo, umístit jej někde a zapojovat k elektrickým rozvodům, ale

ani hledat volný port na přepínači. Velmi snadno lze zjistit stav hostovaných systémů a vytvořit nový stroj včetně instalace.[2] Samozřejmě zrušení nebo přeinstalace virtuálního stroje je opět snazší a výrazně šetří čas obsluhy.

Přestože přínosy virtualizace jsou obecně stejné pro všechny virtualizační nástroje, má Xen jednu velkou výhodu – *migraci*. Pokud jsou pro Xen splněny i některé podmínky z pohledu IT infrastruktury[3], pak vám umožní migrování (přesunutí) virtuálních strojů. Tím lze zvýšit dostupnost hostovaných strojů, protože před plánovanou odstávkou lze virtuální stroje přemigrovat na jiný Xen server.

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

Omezení

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

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

Xen na ZČU

O nasazení nějakého virtualizačního nástroje se začalo uvažovat v polovině roku 2003, kdy se nahromadilo několik požadavků na samostatný testovací stroj umístěný na serverovně. V té době chybělo skoro vše: místo v racku, záložní zdroje a hlavně volné porty na přepínačích. Naštěstí tou samou dobou byl již projekt Xen stabilizován v podobě řady 2.x. V porovnání s jinými virtualizačními nástroji měl vždy navrch, a to jak s cenou, tak rychlostí či dokumentací. Ná-

[2] Automatická instalace virtuálního stroje metodou FAI včetně přípravy konfiguračních souborů pro FAI i Xen zvládáme na ZČU do 10 minut.

[3] Více v části *Migrace*.

sledovalo testování možností Xenu, ale i kompatibility s ostatním vybavením IT prostředí ZČU, zvláště AFS[4], FAI[5] a podpora VLAN (802.1q).

Na základě zkušeností byl zakoupen stroj s potřebným HW vybavením a rozhodnuto, že datové disky virtuálních strojů budou umístěny na Fibre Channel[6]. První rok byly na takto vzniklém stroji provozovány virtuální stroje, kterým by případný pád nehrozil, převážně pro testování. Ovšem jak čas plynul, software určený k testování přecházel v ostrý provoz a s ním narůstala i důležitost zachování provozu Xenu. Po naplnění kapacity se pořídil druhý stroj a život šel dál.

Nyní máme téměř čtyřletou zkušenost s provozem Xenu a směle jej mohu prohlásit za velmi stabilní. Nikdy jsme neřešili SW problém pádu hypervisora a naštěstí ani HW problémy. Nyní máme celkem 4 stroje, z nichž jeden je určen pro provoz ostrých virtuálních systémů, druhý pro testovací virtuální stroje. Třetí stroj byl koupen z grantu fondu Rozvoje CESNET[7] na ověření migrace virtuálních strojů. Konfigurace tří serverů je téměř stejná:

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

Čtvrtý stroj byl také zakoupen v rámci grantu Fondu Rozvoje CESNET[8], který navazoval na předchozí projekt, tentokrát byl směrován k testování 64bitové architektuře a možnosti provozu nemodifikovaných virtuálních strojů při užití HW podpory virtualizace. Stroj je obdobné konfigurace, je ale 64bitový a má podporu hardwarové virtualizace.

Problémy při nasazování

Při zavádění Xenu jsme se potýkali s několika zásadnějšími problémy. Jedna z významných potíží byla kompilace jádra do `.deb` balíků nástrojem `make-kpkg`.

[4] Andrew File System – http://www.openafs.org/

[5] Fully Automated Installation – http://www.informatik.uni-koeln.de/fai/

[6] Fibre Channel – http://hsi.web.cern.ch/HSI/fcs/

[7] Grant je veden pod číslem 154R1/2005, elektronické zdroje použité při řešení jsou k dispozici na http://support.zcu.cz/index.php/CIV:Granty/Overeni_migrace_Xen_virtualnich_stroju

[8] Číslo grantu je 192R2/2006, grant v současnosti probíhá a jeho součástí je také tento příspěvek.

V tomto případě šlo zvláště o nastavení správných parametrů a přejmenování defaultních názvů patchovaných jader. Zprovozněním kompilace přes `make-kpkg` byla také úspěšně zkompilován modul klienta pro OpenAFS do balíčku. V současné době používáme předkompilovaných balíků s podporou Xenu přímo z distribuce Debian.

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

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

Migrace

Jak jste si všimli, byla z popisu vynechány informace o nástrojích a provozu Xenu. Důvodem je nošení dříví do lesa, tato část je totiž velmi dobře zdokumentována na stránkách projektu http://xen.sf.net/ a k dispozici je také LiveCD na kterém si lze Xen vyzkoušet. Mimo jiné existuje mnoho různých howto, které vás do problematiky instalací a provozu uvedou.

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

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

Právě tomuto se říká v pojetí Xenu migrace. Tu rozlišujeme na *off-line* a *on-line*. Rozdíl mezi nimi spočívá v rychlosti migrace a dostupnosti systému během migrace. Off-line migrace probíhá prakticky ve výše popsaných třech fázích: uspání, přesun obrazu paměti na nového hypervisora, probuzení. Během těchto fázích není hostovaný systém dostupný, ale migrace je provedena bez zbytečného zdržování. Naproti tomu on-line migrace umožňuje jen minimální výpadek (méně než 1 vteřina[9]), ale provedení migrace je časově i procesorově náročnější – musí se navíc hlídat zápisové operace do paměti.

[9] Záleží na zatížení stroje, zvláště počtu prováděných operací v paměti.

Nemodifikované operační systémy

Mezi novinky v řadě 3.x patří možnost provozu nemodifikovaných operačních systémů. Má to ale háček, je potřeba mít HW podporu virtualizace. Podporou se rozumí, že musíte mít správný čipset, dále procesor a zapnutou podporu v BIOSu. V Xenu jsou podporovány technologie od Intelu[10] i AMD[11].

Na ZČU se objevila potřeba občas otestovat novou aplikaci v MS Windows. Kupovat a instalovat nový stroj pro tyto občasné potřeby není z ekonomického hlediska ideální.

Podle různých návodů a vlastních zkušeností se nám podařilo na stroji s podporou IntelVT[12] rozběhnout nemodifikované operační systémy. Virtuálnímu stroji jsou potřebná rozhraní k fyzickému stroji emulována, jde zejména o BIOS, grafickou kartu, síťovou kartu a disky. Xen využívá pro tyto účely kód z projektu QEMU, což je patrné při startu. Tato emulace I/O rozhraní může být přímo navázána na jednotlivé komponenty díky řízení přístupu, který zajišťuje hardwarová podpora virtualizace. To nám zajistí dostatečné oddělení virtuálních strojů a zároveň neztrácíme rychlost daných operací. Zároveň není potřeba provozovat modifikovaný systém, který by si rozuměl s hypervisorem v Xenu.

V říjnu 2007 jsme měli k dispozici jeden stroj s hardwarovou podporou virtualizace. Zároveň je tento stroj plně 64bitový, což nám otevřelo cestu ubírat se směrem ke zkoušení nemodifikovaných operačních systémů a poskytnout správcům služeb testovací platformu pro 64bitovou architekturu.

Závěr

Tento příspěvek si nekladl za cíl seznámit uživatele s detaily virtualizace, proto se vyhnul i některým důležitým pojmům nebo je jen okrajově zmínil. Z článku by měl být patrný způsob nasazení a vývoj virtualizace na Západočeské univerzitě, možnosti jeho využití i nástin plánů do budoucnosti. Dnes je k dispozici více virtualizačních technologií, které poskytují takový výkon, aby mohly být nasazeny v provozu. Velmi zdárně se vyvíjí projekt Linux VServer[13] nebo KVM[14], který je od verze 2.6.20 přímo v jádře. Záleží pouze na vašich potřebách, protože každá technologie má něco pozitivního i negativního a bez kompromisů se neobejdete.

[10] Nazývá se kódovým označením *Vanderpool*, procesor lze poznat podle příznaku `vmx`.
[11] AMD ji označuje názvem *Pacifica*, procesory obsahují příznak `vms`.
[12] IntelVT je starší název podpory virtualizace, která nakonec dostala název Vanderpool.
[13] Domovská stránka projektu VServer: http://linux-vserver.org
[14] Informace o projektu KVM jsou na http://kvm.qumranet.com/kvmwiki

Literatura

[1] Miroslav Suchý: *Úvod do virtualizace pomocí XENu*
http://www.root.cz/clanky/uvod-do-virtualizace-pomoci-xenu/

[2] Ian Pratt a kol.: *Live Migration of Virtual Machines*
http://www.cl.cam.ac.uk/research/srg/netos/papers/2005-migration-nsdi-pre.pdf

[3] Kolektiv autorů: *Xen 3.0 User Manual*
http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/user.pdf

Další krok ke konvergenci sítí – ENUM

Pavel Tůma

E-mail: pavel.tuma@nic.cz

Téma konvergence sítí je v oblasti telekomunikací evergreenem posledních let. V nedávné době se objevil nový systém, který může v této oblasti přinést doslova revoluci. Přišla ze světa internetu a jmenuje se ENUM. Zkratka ENUM znamená E164 Number Mapping, tedy mapování telefonních čísel. Pro pochopení principu, jak funguje a čím mění telekomunikační sféru, se vraťme k jednomu ze základních stavebních prvků dnešního internetu.

ENUM a DNS

Doménový systém vznikl proto, že v internetu se adresace provádí pomocí číselných IP adres, které jsou pro člověka špatně zapamatovatelné. Naproti tomu slovní názvy jsou zapamatovatelné dobře a funkcí DNS je zprostředkovat vztah mezi doménovým jménem a jemu odpovídající číselnou IP adresou. Právě této schopnosti přiřazování adres využívá technologie ENUM.

Principem technologie je, že k telefonnímu číslu se přiřadí adresa služby na internetu. Telefonní číslo je nejprve nutné převést pomocí jednoduché transformace na doménové jméno. To se provede tak, že v zápisu čísla v mezinárodním tvaru se vynechají úvodní nuly a všechny nečíselné znaky, poté se číslo otočí a jednotlivé číslice se oddělí tečkami. Na závěr se přidá doména .e614.arpa, což je začátek doménového stromu věnovaného pro ENUM. Z hypotetického telefonního čísla 00 420 123 456 789 získáme doménové jméno 9.8.7.6.5.4.3.2.1.0.2.4.e164.arpa. K tomuto doménovému jménu umístíme na zvolený DNS server adresu libovolné služby (nebo i více služeb) na internetu v běžně používaném zápisu formou URI (Uniform Resource Identifier). Telefonnímu číslu tak lze přiřadit e-mailovou adresu nebo adresu webových stránek. V současnosti má ovšem největší význam a využití přiřazení adresy služby používané pro přenos hlasu po internetu – SIP adresy (případně i H323 adresy).

Záznam v DNS obsahující SIP adresu říká všem, kteří se dokáží na ENUM doménu dotázat, jak se mají s tímto číslem spojit pomocí internetu a služby SIP. Obdobně to funguje i u dalších služeb. ENUM umožňuje do DNS vložit k telefonnímu číslu údaje o více službách najednou a to včetně priorit, která ze služeb je preferovaná více a která méně. Ten, kdo na svůj dotaz obdrží jako

odpověď údaje o více službách, si následně sám může vybrat konkrétní službu, pomocí které bude s uživatelem telefonního čísla dále komunikovat. Uživatel volí ze všech dostupných služeb, případně z omezeného počtu, který vyplývá z technologických možností jeho telefonního zařízení.

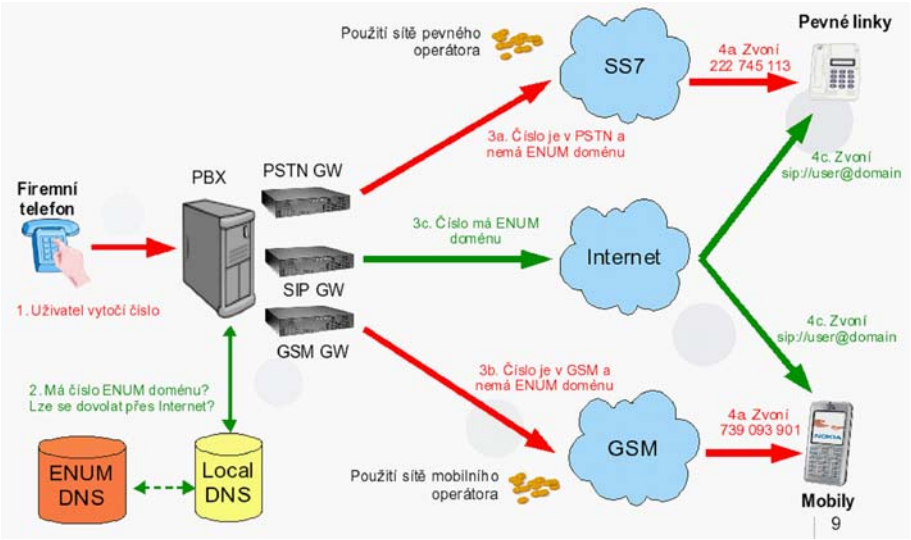
Nyní se vraťme k nejpoužívanějšímu případu – SIP adrese pro spojení se s uživatelem telefonního čísla pomocí VoIP. SIP adresa v ENUM může vypovídat o možnosti spojit se přes VoIP, ovšem za určitých okolností může říkat, v jaké síti kterého telekomunikačního operátora se dané číslo nachází, a jak se hovory, zprávy či jiné telekomunikační služby do této sítě směrují. Záleží na tom, kdo se do ENUM dotazuje. Uživatele telefonního přístroje, který volá na telefonní číslo zajímá, jestli se s číslem lze spojit přes internet, protože za přenos hovoru po internetu volající neplatí; na rozdíl od veřejné telefonní sítě, kde si operátor účtuje minutové poplatky. Telekomunikačního operátora v momentu spojování hovoru svého zákazníka naopak zajímá, do jaké sítě má hovor poslat, a na jaký hraniční prvek cílové sítě hovor předat. Proto existují dva základní typy ENUM. ENUM pro uživatele tzv. UserENUM nebo také Public ENUM a ENUM pro operátory tzv. Infrastructure ENUM.

User ENUM

User ENUM slouží koncovým uživatelům (držitelům) telefonních čísel, tedy těm kdo mají telefon s příslušným číslem na stole či majitelům mobilních telefonů. Ti jediní mají možnost do uživatelské větve ENUM registrovat ENUM domény svých telefonních čísel a zveřejňovat své SIP adresy. Zároveň striktně platí opt-in princip – tj. uživatel, který toto nechce udělat, své číslo registrovat nemusí. Informace vložená do User ENUM se stává veřejnou, neboť kdokoliv může poslat do DNS dotaz na doménu ENUM stejně jako na kteroukoliv jinou internetovou doménu.

Přínos informace z DNS pro volajícího, který se před vlastním spojením hovoru nejdříve dotáže do ENUM, je zřejmý. Pokud existuje možnost spojení hovoru přes internet, může volající v takovém případě ušetřit nemalé částky za hovorné, které by jinak musel platit za spojení hovoru přes běžnou telefonní síť. Uživatel ENUM tak vždy využívá ten nejméně nákladný způsob volání, viz obr. 1.

Možností hovoru zdarma bez ohledu na cílovou síť výčet přínosů systému ENUM zdaleka nekončí. Zejména firemní uživatelé přivítají možnost poskytnout svým zákazníkům službu, kdy se k nim dovolají na běžná telefonní čísla zdarma. Dalším přínosem systému ENUM je ušetření nákladů na provoz bezplatných linek nebo linek se sdílenými náklady tzv. „barevných“ linek. Volající má v případě zelené linky hovor vždy zdarma (nebo za nižší částku u barevných linek), ale náklady na tento hovor hradí provozovatel linky. V případě spojení přes internet je hovor zdarma pro obě strany.



Obr. 1

Aby se ENUM dal plně využít, musí být splněno několik podmínek. Jelikož ENUM je o VoIP telefonování, volající musí být schopen iniciovat spojení hovoru přes internet a volaný musí umět takový hovor přijmout. K tomu postačí použití libovolných IP telefonů na obou stranách hovoru. Na trhu je dnes široká škála přístrojů od stolních, přes přenosné až ke kombinovaným IP a GSM telefonům, VoIP už dávno není pouze záležitostí softwarových telefonů v počítači. Dále se volající musí dotázat v momentu spojování hovoru do DNS na ENUM doménu. To zajistí správně nastavená ústředna, která rozhodne o směrování hovoru. U firem je to pobočková ústředna, v domácnostem to obstará přímo ústředna daného VoIP operátora. Ústředny, které ENUM dotazy podporují, jsou v dnešní době běžně dostupné. Ať už se jedná o open-source řešení ala Asterisk, Open SIP Express router nebo o výrobky renomovaných firem typu Cisco, Kapch, Siemens apod.

V neposlední řadě je zapotřebí mít zaregistrovanou ENUM doménu a vloženou správnou SIP adresu do DNS. Registr ENUM domén pro česká telefonní čísla prefixu 420 spravuje sdružení CZ.NIC, které je i správcem české národní domény .cz. Samotný proces registrace je obdobný jako u registrace domén .cz s jediným rozdílem – tzv. validací. Validace slouží k ověření, že žadatel o registraci domény je skutečně uživatelem registrovaného čísla. Provádí se většinou pomocí jednorázového hesla doručeného na registrované číslo pomocí SMS či hlasového hovoru anebo na základě předložení měsíčního vyúčtování služeb od operátora.

Proces registrace nemusí uživatel absolvovat nutně sám, v optimálním případě to za něj udělá operátor v rámci balíčku služeb, který nabízí. Přestože je User ENUM určen držitelům čísel, má svůj význam i pro telekomunikační operátory.

Telekomunikační operátor totiž může ENUM a jím umožněné hovory zdarma použít jako prodejní argument při získávání nových zákazníků či jako argument pro přechod na VoIP. Hovory zdarma mohou složit také jako retenční faktor zachování stolních telefonů a „pevných“ linek navzdory trendu poslední doby, kdy volající přecházejí k mobilním operátorům a používají pouze mobilní telefon. VoIP a ENUM pomohou operátorům k prodeji souvisejících produktů a služeb – připojení k internetu, upgrade na ENUM podporující infrastrukturu, apod. ENUM dovoluje operátorům nabídnout zákazníkům i zcela nové služby založené na internetovém protokolu IP.

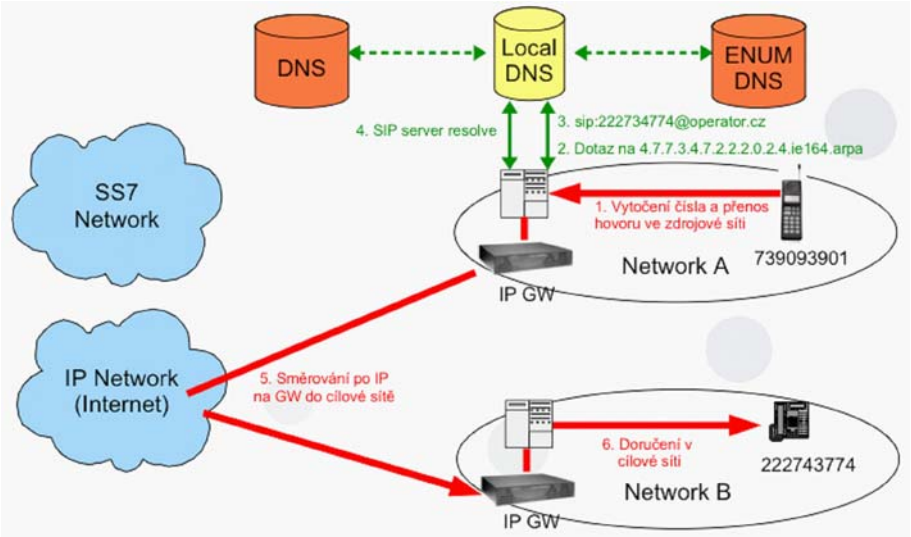
Nejen v České republice, ale i v několika dalších zemích Evropy a ve světě je uživatelský ENUM v plném provozu. Například plný provoz u našich sousedů v Rakousku a Německu je výhodný, neboť do těchto zemí se volá častěji než do vzdálenějších. Právě v sousedních státech sídlí nejvíce poboček firem podnikajících i u nás. Prakticky všechny evropské země již mají delegovanou doménu svého národního prefixu. Mimo Evropu funguje uživatelský ENUM v některých zemích východní Asie – např. v Koreji, Japonsku a Číně.

Infrastructure ENUM

Doménový strom Infrastructure ENUM je umístěn v jiné doméně než .e164.arpa a je zcela oddělen od User ENUM. Infrastructure ENUM je určen telekomunikačním operátorům, kteří do něj registrují domény pro čísla, která mají ve svých sítích. Tím dávají ostatním telekomunikačním operátorům k dispozici informaci o tom, v které konkrétní síti se číslo nachází a zároveň údaj,jak hovor na takové číslo směrovat do správné sítě.

Domény v Infrastructure ENUM by měly pokrývat všechna čísla, která má příslušný operátor přidělena od regulátora (v našem případě od Českého telekomunikačního úřadu), protože je nutné zajistit ochranu údajů před zneužitím. Pokud by byla registrována pouze aktivní čísla, bylo by je možné zjistit metodou hrubé síly tj. postupným dotazováním na všechna možná čísla (v číslovacím plánu ČR je teoreticky možné mít jednu miliardu čísel, v praxi je to ovšem mnohem méně) v relativně krátkém čase a pomocí velmi malých prostředků. Neznamená to, že je nutné mít stejný počet domén jako telefonních čísel, protože při těchto počtech by to bylo technicky obtížně zvládnutelné. Operátoři vloží do registru pomocí jedné domény celé číselné řady. Tyto bloky se budou rozpadat na menší bloky v případě, že je nějaké číslo z bloku přeneseno do jiné sítě. Přesto počet záznamů bude výrazně menší a stejně tak nároky na technické řešení registru.

Konkrétní využití Infrastructure ENUM je obdobné jako v případě User ENUM. Před spojením hovoru se na straně operátora musí vždy provést dotaz do registru a zjistit cílovou síť hovoru (část SIP adresy za zavináčem v SIP adrese). Dalšími dotazy do DNS zjistit SIP server, který danému jménu odpovídá, a konečně se s tímto serverem spojit a předat mu hovor pro konkrétního uživatele (část SIP adresy před zavináčem). Celý postup ilustruje obr. 2.



Obr. 2

Operátoři mohou tímto způsobem vyřešit několik problémů naráz. Asi největším přínosem pro ně bude propojení jejich IP sítí – IP peering. Další výhodou je snížení složitosti systémů. Dnes musí ústředny operátorů získávat informace pro směrování hovorů z několika různých zdrojů (databáze přenositelnosti čísla, statických směrovacích tabulek a dalších systémů). Při použití ENUM získají vše z jednoho místa. A nemusí jít jen o směrovací informace. Do DNS je k telefonnímu číslu možné vložit de facto jakoukoliv další informaci, která operátora zajímá. Navíc k těmto výhodám přidejme i všechny další plynoucí z použití DNS, které rozhodně nejsou zanedbatelné, protože snižují náklady, údržbu a správu systémů. Záznamy v ENUM totiž udržuje každý pouze o sobě samém, nicméně z principu DNS mají všichni vždy právě aktuální informaci o všech ostatních. Vynikající vlastností systému DNS je jeho velmi dobrá škálovatelnost a rychlost bez ohledu na počet záznamů, které jsou v něm uloženy nebo na počet dotazů, které do něj proudí. I při dnešních miliónech počítačů připojených k internetu je systém schopen odpovědět na dotaz v řádech milisekund. Díky snadnému řešení

redundance je systém DNS velmi spolehlivý. Za celou dobu jeho existence (první DNS standard pochází z osmdesátých let minulého století) neměl systém žádný výpadek!

Infrastructure ENUM je zatím spíše otázkou budoucnosti. A to díky dvěma faktorům: operátorský ENUM na rozdíl od uživatelského není zatím standardizován a co je neméně důležité – vyžaduje dohodu mezi operátory o tom, jak bude celý systém v dané zemi fungovat a to po stránce technické i administrativní. Postupně tak budou vznikat jakési ostrůvky – menší „federace“ operátorů, kteří budou používat nějakou formu Infrastructure ENUM mezi sebou. Očekává se, že se tyto privátní ostrůvky budou postupně spojovat a že jednou vznikne stejně globální systém, jako je dnes k dispozici v User ENUM.

Obecný ENUM

Při pohledu do budoucnosti se navíc v případě ENUM nemusíme vůbec omezovat na telekomunikační svět a ukládání telefonních čísel. ENUM totiž není nic jiného než jakási databáze, kde se vyhledává podle klíče, kterým je nějaké strukturované číslo, a odpovědí je seznam údajů. Pokud zapátráte v paměti, s jakými strukturovanými čísly se běžně setkáváte, zjistíte, že jich není zrovna málo – čísla čárových kódů, státní poznávací značky, poštovní směrovací čísla, rodná čísla, ISBN, RFID tagy a mnoho dalších. Není těžké a vzdálené od reality si představit například následující situaci: výrobce, který vyváží své výrobky osazené RFID tagy do zahraničí, umístí do svého DNS záznamy k RFID tagům s údaji k jednotlivým výrobkům. Celník, který na hranici kontroluje kamión převážející tyto výrobky, pomocí RFID čtečky přečte RFID tagy, pošle dotaz do DNS a má okamžitě k dispozici informaci o tom, o jaké zboží se jedná, odkud pochází apod. Další uživatelem informace může být prodejce, který si zjistí datum výroby či technologické údaje od výrobce. I sám zákazník získá dotazem do DNS na příslušný RFID tag odkaz na uživatelský manuál nebo na často kladené dotazy.

ENUM má díky výše popsaným výhodám DNS šanci stát se databází i pro další strukturovaná čísla.

Plus DNS oproti jiným databázovým systémům tkví v tom, že DNS je globální, otevřený – každý si může do DNS uložit nějakou informaci a každý do něj může poslat dotaz, a v neposlední řadě je velmi jednoduché jej použít.

ENUM není žádný zázračný vynález, což ovšem nijak nesnižuje jeho význam; minimálně do oblasti telekomunikace přináší čerstvý vítr a změnu. Nevíme zatím jak velká to změna bude, ale cesta už byla nastoupena...

DNSSEC

Ondřej Surý

E-mail: ondrej@sury.org

Stručná historie DNS

DNS (Domain Name System) je hierarchický systém doménových jmen, který slouží k převodům jmenných názvů (doménových jmen) na IP adresy počítačů a serverů (A a AAAA záznamy), a naopak (PTR záznamy). Mezi další funkce patří např. poskytování informací o směrování pošty (MX záznamy), o směrování IP telefonie (ENUM) a další funkce. DNS je tedy decentralizovaná distribuovaná databáze síťových informací, jehož protokol obsluhují DNS servery. Překlad jmenných názvů na IP adresy však nebyl vždy zajišťován protokolem DNS.

Systém pojmenování počítačů textovými názvy vznikl velmi záhy po vzniku sítě ARPANET, což byla síť, která se později stala základem toho, čemu dnes říkáme internet. Adresace počítačů pomocí číselných adres byla velmi nepraktická a proto se systém velmi rychle ujal.

V úplných počátcích žádný systém neexistoval, každý počítač si udržoval svou vlastní kopii souboru *hosts* pro překlad jmen na číselné adresy. Tento soubor byl spravovaný ručně a záznamy do něj přidával člověk, operátor. Jak počítače připojené k síti přibývaly, tak začínalo být jasné, že je potřeba systém sjednotit. První zmínku o této změně je možné najít v RFC 606 (pozn. autora: zkratku RFC dobře popisuje také česká verze Wikipedie) z prosince 1973, kde je popsán návrh centralizované správy souboru *hosts* na jednom soustředěném místě. Všechny počítače si pak z tohoto centrálního místa v pravidelných intervalech stahovaly aktuální verzi souboru se jmény počítačů. V této podobě fungoval systém do roku 1983.

S postupem času a nárůstem připojených počítačů začalo být počátkem osmdesátých let jasné, že centralizovaný systém je pro správu většího množství záznamů nevhodný. Výsledek probíhající diskuze vyústil v sérií dokumentů RFC 881 až 883, kde John Postel a Paul Mockapetris vytyčují základy současného systému DNS. Na základě těchto RFC dokumentů Paul Mockapetris píše úplně první implementaci DNS server na světě pojmenovanou Jeeves.

O tři roky později sepisuje Paul Mockapetris dokumenty RFC 1034 a 1035, které shrnují dosavadní praxi a vývoj na poli DNS. Nutno dodat, že tyto dva

dokumenty stále obsahují základní definice systému DNS a ač byly mnohokrát aktualizovány, tak jsou stále platné. V roce 1988 přichází na půdě univerzity v Berkeley (UCB) na svět první verze dnes stále nejpoužívanějšího DNS serveru Bind a to až do verze 4.8.3. Vývoj verze 4.9 krátce sponzorovala firma DEC a od verze 4.9.3 až dodnes se o vývoj stará sdružení ISC. Ale to již trochu předbíháme, takže se vraťme zpátky do roku 1990.

V roce 1990 byl internet stále poklidné místo, kde jeho uživatelé žili spokojeně a ve vzájemné harmonii. Nebo ne? V tom samém roce přichází Steven M. Bellovin na zásadní chybu při používání DNS. Vzájemná důvěra mezi počítači byla v té době většinou řešena pouze na základě správného doménového jména v DNS. V zásadě se nejedná o chybu samotného DNS, ale chybu přílišné důvěry v systém DNS. Publikace této práce byla odložena z bezpečnostních důvodů o celé čtyři roky, ale jak sám autor poznamenává, nebylo to příliš moudré rozhodnutí, protože informace z tohoto dokumentu stejně prosákly na veřejnost a byly zaznamenány útoky, které využívaly tento nadbytek důvěry v doménový systém.

Po zveřejnění tohoto nedostatku se na půdě IETF, což je organizace, která stojí za většinou standardů používaných na internetu, začíná uvažovat nad zabezpečením systému DNS. Mezitím Eugene Kashpureff objevuje další zranitelnost v současných implementacích DNS serverů (tedy v té době především serveru Bind), která v DNS umožňuje podvrhnout libovolný záznam. V roce 1997 pak používá tuto chybu k celosvětovému nebo spíše celointernetovému přesměrování stránek registrátora InterNIC na stránky své firmy AlterNIC. Ještě v témže roce vzniká první dokument popisující kryptografické zabezpečení systému DNS – RFC 2065. Tento dokument je pak během dvou dalších let rozpracován a v roce 1999 je v RFC 2535 publikována první verze systému DNSSEC. DNSSEC definuje do systému DNS nové záznamy, které kryptograficky zajišťují integritu dat poskytovanou DNS servery. Pomocí DNSSEC je možné zjistit, zda informace, které uživatel dostane ze systému DNS, nebyly změněny. Jedná se o podobný princip jako u elektronického podpisu; data mají kryptograficky vytvořenou obálku, pomocí které lze ověřit, že obsah nebyl modifikován žádnou třetí stranou.

První implementace DNSSECu dle RFC 2535 je připravena v DNS serveru Bind, bohužel další dva roky nasazení DNSSECu stagnuje a v roce 2001 je vypracována studie, která konstatuje, že tato první verze systému DNSSEC je nevhodná k nasazení, protože libovolná výměna podepisovacího klíče vyžaduje jednak komplexní komunikaci s nadřazeným DNS serverem a jednak se tato změna musí promítnout na všech podřízených DNS serverech. Tyto požadavky tuto verzi DNSSECu odsunuly na vedlejší kolej, začíná se pracovat na verzi nové pracovně nazývané DNSSECbis. DNSSECbis definuje úplně nové záznamy, které nejsou kompatibilní s původní verzí DNSSECu a rozšiřuje DNS o nový druh záznamu – DS, který má výrazně zjednodušit komunikaci s nadřazeným serverem.

V průběhu let 2002 až 2003 je tato nová verze DNSSECu implementována v DNS serveru Bind a ukazuje se, že je na rozdíl od své předchůdkyně životaschopná. V roce 2004 je podpora DNSSECbis implementována ve dvou nezávislých DNS serverech (Bind a NSD 2) a čeká se jen na standardizaci. Ta proběhne až v roce 2005, kdy jsou v březnu vydány dokumenty RFC 4033 až 4035. V říjnu 2005, tedy jen o pár měsíců později, implementuje DNSSEC první TLD doména – švédská .se.

Ani DNSSECbis se ovšem neobešel bez problémů. V návrhu protokolu se počítá s tím, že je zapotřebí mít i kryptograficky ověřené odpovědi o neexistenci určitého záznamu. Z tohoto důvodu vznikl DNS záznam NSEC, který pro určité doménové jméno říká, který záznam jej abecedně následuje. V případě dotazu na neexistující záznam DNS server vrátí jména, která jsou „okolo“, tato odpověď je kryptograficky podepsána. Tento princip ovšem otvírá jeden nepříjemný důsledek – tímto způsobem se dá projít celý doménový prostor, jedná se tzv. zone walk. Návrh řešení existuje v dokumentu RFC 5155 schváleném letos na jaře, který definuje nový druh záznamu NSEC3. Zde je ukryta informace o dalším záznamu.

Dnes se píše rok 2008 a pomocí systému DNSSEC jsou chráněny pouze čtyři národní domény (.se, .bg, .pr a .br), brzy se k nim připojí i doména česká .cz. Nicméně jistý posun lze pozorovat i u domén generických – nedávno vydané nařízení administrativy Spojených států ukládá správci domény .gov, aby do konce ledna 2009 podepsal doménu, kterou používají vládní úřady a organizace ve Spojených státech. V souvislosti s nedávno zveřejněným útokem na DNS, který poprvé odhalil bezpečnostní výzkumník Dan Kaminsky, je nasazení DNSSECu správnou volbou.

Tento text vyšel v loňském roce v říjnu v časopise Connect.

Benefity a úskalí plošného souvislého sledování IP provozu na bázi toků při řešení bezpečnostních hlášení

Tomáš Košňar

E-mail: kosnar@cesnet.cz

Abstrakt

Příspěvek rozebírá možnosti využití plošného a souvislého sledování IP provozu na bázi toků jako přímé podpory při řešení bezpečnostních hlášení. Východiskem jsou praktické zkušenosti s vývojem a provozováním systémů pro zpracování informací o IP provozu a spolupráce se správci sítí a bezpečnostními týmy v prostředí národní počítačové sítě pro vědu, výzkum, vývoj a vzdělávání CESNET2.

Úvod

Sledování IP provozu na bázi toků je v posledních letech stále populárnější monitorovací metoda. Je založena na sběru a zpracování agregovaných informací o IP tocích. Tyto informace o provozu lze získávat například z aktivních síťových zařízení, specializovaných HW sond, či sond založených na běžném počítači s nainstalovaným specifickým programovým vybavením. Koncept podle kterého jsou tyto informace vytvářeny a přenášeny do míst zpracování byl formulován okolo roku 1996 vývojáři Cisco Systems pod názvem NetFlow. Příspěvek je zaměřen na zhodnocení využití těchto informací o provozu jako podpory pro řešení bezpečnostních hlášení. Východiskem jsou praktické zkušenosti s návrhem, vývojem a provozováním systémů pro zpracování těchto informací v prostředí páteře sítě CESNET2 – národní počítačové sítě pro vědu, výzkum, vývoj a vzdělávání.

Požadavky plynoucí z bezpečnostních hlášení

Požadavků směřujících do oblasti analýzy provozu sítě je celá řada, ale v kontextu s řešením bezpečnostních hlášení se jedná prakticky o dvě základní skupiny dotazů mající za cíl:

1. potvrdit nebo vyvrátit fakta specifikovaná v bezpečnostním hlášení v míře, v jaké je to na úrovni informací o provozu možné.

   Toto se týká především hlášení s jasně vymezenou událostí související s konkrétní síťovou komunikací. Zpravidla se jedná o hlášení mající příčinu např. v porušení užívacích práv nebo v cílených systematických síťových útocích (DoS, DDoS, „network scanninng“, slovníkové útoky, TCP SYN flooding atd.), phishingu apod. Relevantní komunikaci je nutné pomocí analýzy provozu potvrdit. V některých případech však také vyvrátit. A to je možné za předpokladu, že máme k dispozici dostatečně přesné provozní informace o vlastní síti, jde o hlášení o jednosměrném útoku z naší IP sítě a zároveň není principiálně vyloučeno, že si zdrojové IP adresy mohl z nám přiděleného IP rozsahu „vypůjčit“ útočící uzel z úplně jiné sítě. Vzhledem k tomu, že v globálním Internetu nejsou striktně aplikovány politiky směrování podle zdrojových adres je takové schéma útoků naprosto běžné.

2. zkusit na základě indicií specifikovaných v hlášení nalézt na úrovni provozních informací relevantní informace, včetně plošného rozsahu a vymezení v čase.

   Zde je východiskem pro analýzu provozu zpravidla rámcové vymezení směřující k jednomu nebo více zdrojům nebo cílům komunikace. Požadavkem směřující do oblasti analýzy provozu je potom zjistit, zda se v daném období „chovaly uzly sítě normálně“ nebo ne. A pokud ne, tak jakým způsobem se jejich komunikace vymykala normálu. Taková situace nastane například při podezření na kompromitování uzlu sítě a snaze zjistit odkud a jakým způsobem komunikace byl uzel kompromitován, případně odkud je aktuálně pravděpodobně využíván.

Vzhledem k výše uvedeným typickým požadavkům na analýzu provozu je jasné následující: pro hodnotnou podporu musíme zajistit maximálně věrný obraz o provozu sítě – tedy maximální možné množství provozních informací. Tyto informace musíme být schopni uchovávat po nezbytně nutnou dobu a mít k dispozici dostatečně výkonný a komplexní vyhledávací aparát. To vše za přijatelného poměru cena/výkon.

Provozní informace na bázi IP toků

Provozní informace nebo lépe záznamy o IP provozu se skládají z informací vyňatých ze sítí přenášených datových bloků – paketů. Jedná se převážně o informace z hlaviček IP datagramů (resp. i z následujících hlaviček protokolů z rodiny TCP/IP). Ty mohou být rozšířeny o další související informace, a to v závislosti na místě vzniku a jeho dalších vlastnostech (např. čísla autonomních systémů na

směrovačích – jsou-li k dispozici BGP tabulky nebo např. SNMP indexy vstupního a výstupního síťového rozhraní), případně o informace svázané s technologií nižších vrstev (např. MAC adresy). Takto vzniklý záznam o provozu je určitou dobu držen ve vyrovnávací paměti a po tuto dobu je upravován informacemi vyňatými z následujících přenášených paktů příslušných danému toku. Postupně tak vzniká agregovaná informace o provozu.

Záznam o provozu sestává přibližně ze tří skupin informací – identifikátorů toku, objemových informací a příznaků toku. Identifikátory toku tvoří klíčové údaje jednoznačně identifikující tok (např. zdrojová a cílová IP adresa, protokol, čísla portů) – všechny přenášené pakety příslušné danému toku mají tyto informace stejné a tato část provozního záznamu se s dalšími zpracovávanými pakety téhož toku nemění. Objemové informace jsou naopak měněny s každým paketem příslušným danému toku – jedná se o celkový objem toku, celkové množství paketů tvořících tok nebo časy příchodu prvního a posledního paketu toku. Příznaky toku obsahují rozšiřující informace o přenosu jako jsou např. TOS bity (Type of Service) z IP hlaviček nebo TCP vlaječky (TCP flags). Tato část provozního záznamu obsahuje všechny hodnoty příznaku které se v toku objevily (funkce OR při úpravě záznamu na jednotlivých bitových pozicích).

Tím, že provozní záznam vzniká z informací z prvního zaznamenaného paketu a je upravován informacemi z paketů, které mají stejný klíč – tj. identifikátory toku, jedná se o záznam o přenosu dat v jednom směru.

Po expiraci, která je zpravidla dána konfigurací celého mechanismu nebo je vynucena například nedostatkem zdrojů (plná vyrovnávací paměť) je provozní záznam odeslán na jednoho nebo více míst – na tzv. kolektory, kde může být dále libovolně zpracováván. Přenos provozních záznamů je zpravidla realizován jedním z definovaných exportních formátů pomocí UDP protokolu.

Z hlediska zachování soukromí uživatelů je výše popsaný princip optimální, neboť celý mechanismus pracuje pouze s transportními identifikátory – uživatelská data jsou ignorována.

Zdroje provozních záznamů

Jak bylo uvedeno v úvodu, zdrojů provozních záznamů může být mnoho, nicméně pro potřeby důkladně provozní analýzy se zaměříme na dvě typové skupiny zdrojů – zdroje mající charakter „uzlu“ sítě a zdroje mající charakter sondy na lince. Jaké jsou jejich výhody a nevýhody:

1. Zdroje charakteru „uzel“ sítě jsou zpravidla směrovače, tedy aktivní síťová zařízení, která sama zajišťují distribuci IP provozu do jednotlivých směrů. Mechanismus generování provozních záznamů je přidanou hodnotou k základním funkcím. Výhodou je, že generované provozní záznamy mohou

pokrýt všechny směry potenciálního přenosu IP provozu (tj. všechna existující rozhraní směrovače) a zároveň obsahují informace o tom, ze kterého vstupního do kterého výstupního síťového rozhraní zařízení byl příslušný tok směřován. Takto lze tedy získat provozní informace o celém topologickém celku. Naopak nevýhodou jsou zvýšené nároky na zdroje (procesor, paměť), které lze v případě nouze korigovat konfigurací, byť za cenu ztráty celkové vypovídací hodnoty (každopádně se při rozumném postupu lze vždy vyhnout potenciální nestabilitě zařízení jako celku). Také při potřebě analyzovat IP provoz v čisté L2 infrastruktuře je nasazení směrovače pouze kvůli systematickému generování provozních záznamů poměrně absurdní. Další omezení vyplývají i z toho, že tuto funkcionalitu implementují pouze někteří výrobci. V neposlední řadě je třeba za přidanou hodnotu připlatit – v závislosti od výrobce např. za licenci k příslušnému modulu operačního systému nebo za HW komponent.

2. Zdroje záznamů o IP provozu charakteru „sonda“ pokryjí v základní podobě pouze jednu linku. Sonda může mít samozřejmě větší množství síťových rozhraní, ale v tom případě musíme být schopni k sondě přivést všechny linky nebo jejich rozbočky, případně zrcadla jejich provozu, což celou záležitost činí složitější a může ovlivnit základní vlastnosti sítě (např. zvýšený útlum apod.). V tomto případě také nemáme k dispozici informace o přesměrování na aktivním síťovém zařízení. Naopak získáváme flexibilitu z hlediska umístění – nejsme-li omezeni technologicky, lze sondu nasadit kamkoli (viz například zmiňovaná L2 infrastruktura). Sonda také nepřináší zvýšené nároky na zdroje aktivních síťových zařízení (s výjimkou požadavků na zrcadlení provozu nebo kompenzace útlumu při rozbočení). V neposlední řadě také sonda něco stojí.

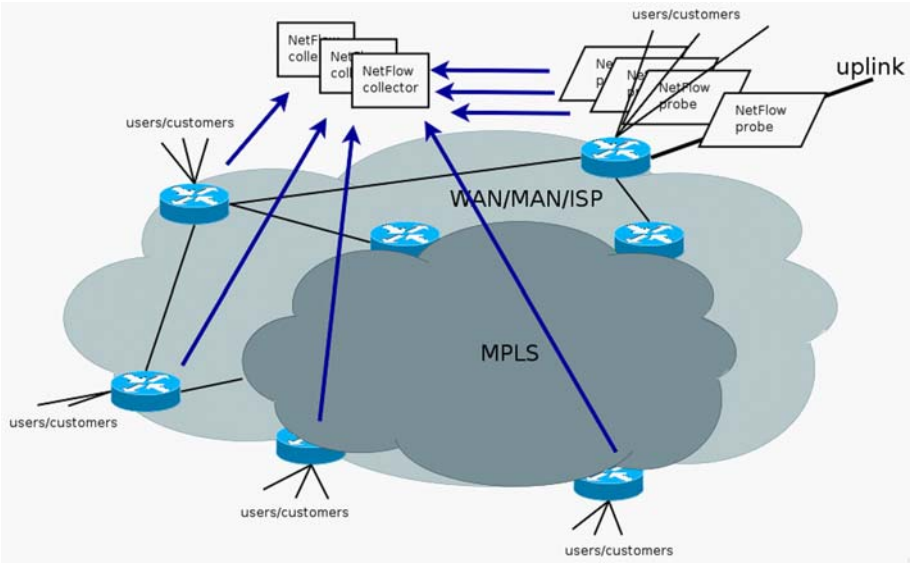
Rozmístění zdrojů provozních záznamů aneb plošné vs. izolované sledování provozu sítě

Při úvahách o analýze provozu platí elementární pravidlo: o provozu, který neteče přes zdroj záznamů se nic nedozvíme.

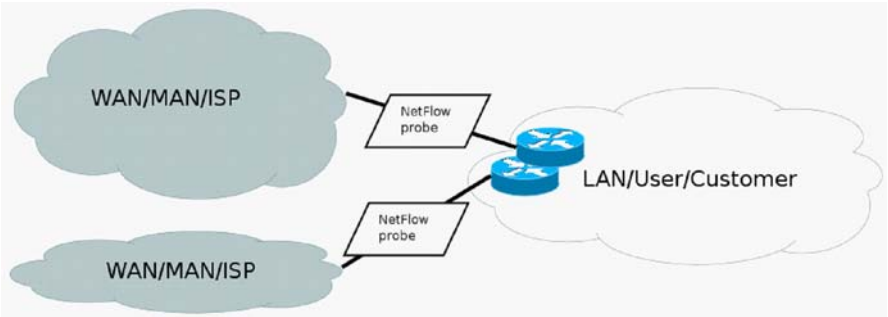
Takže musíme vyřešit otázku, zda pokrýt izolovaně konkrétní vybrané a v tomto smyslu významné části sítě (např. „uplink“, přípojku WiFi infrastruktury, připojení serverových farem apod.) nebo plošně celý síťový celek (např. kompletní hranici MPLS páteře, všechny přípojky do všech open-space kanceláří apod.). Zvolené řešení vždy závisí na konkrétních potřebách, výchozím stavu a finančních možnostech. A rozhodně neplatí, že více znamená vždy lépe. Pro menší koncové sítě zpravidla vystačíme s izolovaným sledováním provozu ve vybraných bodech. Naopak v případě sítí, které mají alespoň zčásti tranzitní

charakter se osvědčuje plošná metoda. Bezesporu výhodou plošného sledování je fakt, že provoz, který sítí prochází, projde zpravidla více než jedním zdrojem provozních záznamů, což výrazně zvyšuje pravděpodobnost, že i při aplikaci vzorkovacích mechanismů (viz níže) budeme mít o hledaném provozu informaci alespoň z jednoho zdroje.

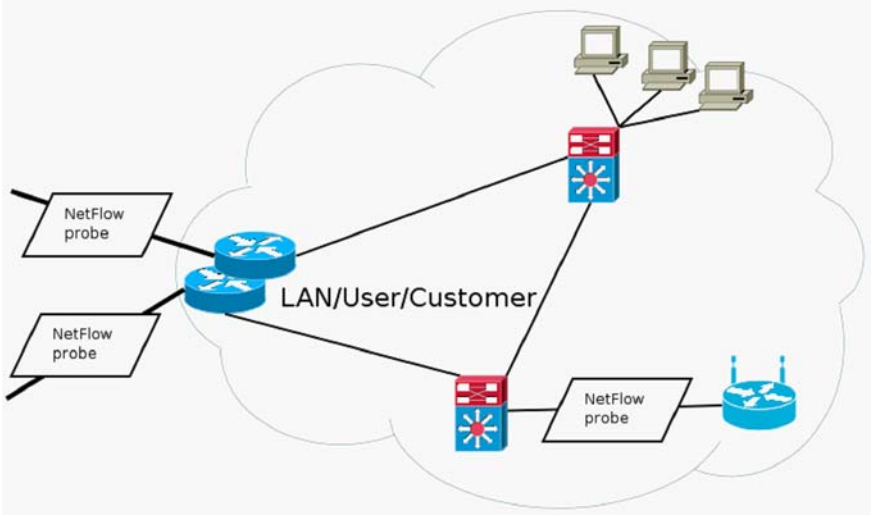
Příklad rozmístění zdrojů provozních záznamů obou typů pro plošný monitoring v topologii sítě je na obrázku 1. V tomto příkladu je naznačeno použití obou typů zdrojů provozních záznamů – sond (NetFlow probe) i směrovačů



Obr. 1 Příklad architektury pro plošné sledování IP provozu sítě



Obr. 2 Příklad architektury pro izolované sledování provozu sítě – externí propojení sítě



Obr. 3 Příklad architektury pro izolované sledování provozu sítě – přípojky k nadřazeným síťovým celkům

včetně naznačení exportu dat na místa zpracování (NetFlow collector). Příklady architektury izolovaného sledování provozu jsou na obrázku 2 – zde se jedná o sledování externích přípojů sítě a na obrázku 3 – s rozšířením na citlivou část infrastruktury – např. bezdrátovou síť.

Množství generovaných provozních záznamů

Množství generovaných a především následně emitovaných provozních záznamů je podstatný parametr podmiňující průchodnost mechanismu následného zpracování provozních záznamů na kolektorech.

Množství provozních záznamů závisí na několika faktorech. Je zde jednoznačná souvislost s objemem provozu. Ale vzhledem k výše popsanému mechanismu vzniku těchto záznamů má zásadní vliv struktura provozu. Demonstrujme to na následujících příkladech:

1. V prvním příkladu vyjděme z předpokladu, že uživatel stahuje z webového serveru s adresou 10.0.0.1 soubor o velikosti 4 GB na stanici s adresou 10.0.0.128 a my sledujeme směr provozu od zdroje k počítači uživatele. Přenosový protokol je TCP, zdrojový port 80 a cílový například 5 126. Předpokládejme, že stroje jsou propojeny technologií Ethernet 1 Gbps. Při standardním chování a MTU bude mít jeden přenesený IP datagram cca 1,5 KB a přenos souboru se rozloží do cca 2,7 miliónu datagramů. Celý pře-

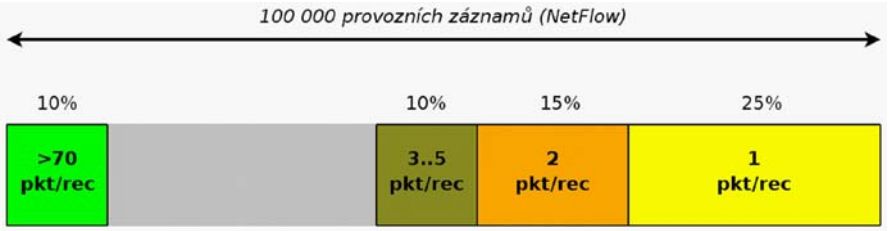
nos bude teoreticky trvat cca 40 vteřin při téměř plném využití přenosové kapacity. Pokud byl pro zdroj provozních záznamů nakonfigurován interval pro expiraci záznamů (a tedy dobu agregace) například 60 vteřin, tak výsledkem pro přenos směrem k uživatelské stanici bude jeden jediný provozní záznam, přestože bylo přeneseno přes 2,5 miliónu datagramů. A to proto, že klíčové identifikátory záznamu (adresy, protokol, čísla portů) byly po celou dobu stejné.

2. Ve druhém příkladu předpokládejme, že tentýž uživatel, ze stejné stanice provede vertikální UDP scan téhož serveru a my budeme sledovat směr přenosu směrem k serveru. Předpokládejme UDP scan s rozsahem portů 1–60 000 a jedním paketem délky 100 B na každý port (včetně protokolární obálky). To znamená, že celkem bude odesláno směrem k serveru cca 6 MB dat v 60 000 datagramech. Celková doba trvání přenosu může být teoreticky pod 1 vteřinu. Výsledkem bude 60 000 provozních záznamů (každý bude reprezentovat jeden přenesený datagram), protože v rámci UDP scanu se měnil s každým odeslaným paketem jeden z klíčových identifikátorů toku – cílový port.

Z předchozích příkladů vyplývá, že pro návrh celého systému pro zpracování provozních záznamů je třeba vzít v úvahu jak jakousi průměrnou rychlost toku provozních záznamů, tak krátkodobé špičky, které mohou vznikat třeba tak, jak je uvedeno ve druhém příkladu.

Poměr počtu emitovaných provozních záznamů vůči počtu přenesených IP datagramů může v závislosti na uživatelském chování, politice přenosu dat a různým dalším omezením výrazně oscilovat síť od sítě. V případě sítě CESNET2 se průměrně pohybujeme v rozsahu 70–100 datagramů na jeden provozní záznam. Přičemž maxima dosahují záznamy s desítkami tisíc datagramů. Naopak minima 1–2 datagramy na záznam reprezentují výrazné procento celkem emitovaných dat (vše je samozřejmě podmíněno i parametry nastavení celé soustavy).

Na obrázku 4 je ukázka rozložení četnosti paketů na záznam (pkt/rec) tak, jak byla opakovaně naměřena v IP páteři sítě CESNET2.



Obr. 4 Příklad rozložení četnosti IP paketů na provozní záznam

Opakovaně je nutno podotknout, že výsledky jsou vždy závislé na konkrétních podmínkách. Nicméně v tomto případě (prezentovaná data odpovídají střední hodnotě několika náhodně vybraných vzorků z období provozních špiček; vzorky vykazovaly zanedbatelný rozptyl) je zajímavé následující. Průměrná četnost počtu IP paketů na 1 záznam byla v tomto případě 70. Ale této průměrné hodnoty dosáhlo pouze 10 % záznamů ze vzorku. Naopak 40 % vzorku tvořily záznamy s velmi nízkou četností paketů (1–2). Zdroj provozních záznamů měl v tomto případě nastaveno vzorkování (viz dále) na úrovni paketů na hodnotě 1, kolektor (příjemce provozních záznamů) měl nastaveno vzorkování na úrovni příchozích záznamů na hodnotě 5. Zároveň byly nastaveny velmi krátké časy expirace záznamů (jednotky vteřin) na zdroji. Při hlubší analýze se ukázalo, že toto rozložení je to způsobeno dvěma faktory – významným podílem „provozního smetí“ (nebo lépe „nevyžádaného provozu“) a předčasným vynuceným exportem záznamů ze zdroje (nedostatek vyrovnávací paměti).

Vzorkování

Běžně se vyskytují případy, kdy některá část celé soustavy nemá potřebný výkon ke zpracování všech příchozích dat – zdroj provozních záznamů nevyjímaje. A vzhledem k tomu, že absolutní množství dat ke zpracování v celé soustavě je dáno parametry provozu, což je z hlediska této soustavy parametr neovlivnitelný, je třeba mít k dispozici mechanismy, které umožní takové situace řešit. Univerzální metodou je vzorkování dat. V celém řetězci pro zpracování provozních záznamů je několik míst, kde lze a zpravidla je implementován vzorkovací mechanismus.

1. Na vstupu zdroje provozních záznamů je většinou (jsou i výjimky) k dispozici možnost nastavit vzorkování na paketové úrovni. To nám umožňuje poslat ke zpracování například pouze každý N-tý přenášený paket. V některých implementacích převážně na jádrových směrovačích se dokonce nelze vzorkování vyhnout a nejmenší možná nastavitelná hodnota je např. 1 : 10.

2. Na výstupu ze zdroje provozních záznamů lze v závislosti na implementaci nastavit například emulaci paketového vzorkování, kdy je u záznamů určených k odeslání porovnáván počet paketů které reprezentují s hodnotou vzorkování. Výsledkem je to, že některé záznamy nejsou odeslány, u některých mohou být modifikovány objemové hodnoty.

3. Na vstupu kolektoru bývají podle konkrétního systému následného zpracování implementovány vzorkovací mechanismy zpravidla na úrovni záznamů o provozu.

Vzorkováním dochází ke ztrátě vypovídací hodnoty. V kontextu analýzy provozu z důvodů bezpečnostních hlášení je to citlivá ztráta. V této souvislosti získává na významu architektura plošného sledování provozu, kdy s každým dalším zdrojem dat, přes který principiálně musel hledaný provoz téci, roste pravděpodobnost nalezení záznamu o jeho uskutečnění. Dalším parametrem, který může pomoci je míra agresivity pro export provozních záznamů ze zdrojů.

Snižováním doby, po kterou je záznam o daném toku držen ve vyrovnávací paměti zdroje (a může být upravován dalšími pakety daného toku), docílíme rozložení delších toků do více provozních záznamů a zvýšíme tak pravděpodobnost jejich zachycení i při nenulovém vzorkování. Celkové množství emitovaných záznamů se zvýší pouze zanedbatelně, neboť dlouhodobé toky tvoří pouze velmi malé procento všech záznamů. Takové nastavení se zhodnotí například při analýze specifických průniků komunikačně neagresivního charakteru nebo při analýze provozu již kompromitovaných zdrojů, které vykazují jakoby běžné, ustálené „síťové chování“.

V této souvislosti je zajímavé alespoň okrajově zmínit důsledky použití různých způsobů vzorkování při aplikaci statistických metod pro detekci provozních anomálií (zpravidla detekce útoků agresivního typu založená na odchylce statistické hodnoty toku od průměrné statistické hodnoty provozu), kdy při stejném nastavení dochází k zásadním rozdílům ve výsledcích při vzorkování na úrovni paketů na vstupu do zdroje provozních záznamů a při vzorkování na úrovni exportovaných záznamů v místě zpracování. Pokud bychom aplikovali zpětně např. vzorkování IP paketů hodnoty 5 na rozložení uvedené na obrázku 4, tak se rozložení změní přibližně následujícím způsobem. Záznamy s více než 5ti pakety zůstanou teoreticky zachovány, počet záznamů s 1 paketem klesne 5×, počet záznamů se 2 pakety klesne na 2/5 původního počtu atd. Podstatné je, že se významným způsobem změní proporce rozložení ve prospěch „stabilního“ provozu a zásadně sníží hodnota maximálních odchylek toků od statistického středu provozu.

Nástroj pro zpracování provozních informací na bázi IP toků

Jak již bylo řečeno výše, pro analýzu provozu na základě bezpečnostních hlášení je třeba mít k dispozici nástroj, který je schopen komplexního prohledávání v provozních záznamech. To samo o sobě nestačí. Bezpečnostní hlášení se týkají provozu, který byl uskutečněn v minulosti, takže součástí takového nástroje musí být efektivní mechanismus ukládání a skladování dat po dobu nezbytně nutnou pro případnou analýzu. Kromě efektivního uchovávání dat a vyhledávacích mechanismů musí být k dispozici i agregační nástroje umožňující provádět

nad množinou nalezených dat třídění agregátů podle rozličných kritérií – tyto vlastnosti jsou nutné především pro nalezení relevantních zdrojů provozních záznamů pro druhou typovou skupinu požadavků bezpečnostních hlášení směrem k analýze provozu (viz výše). Z hlediska výkonu musí takový systém splňovat následující podmínky:

1. Schopnost synchronně zpracovávat a ukládat příchozí provozní záznamy v objemu, který je dán charakterem provozu sítě nebo, který je nezbytně nutný (tzn. že je aplikována přijatelná míra vzorkování) k tomu, aby bylo možné provozní analýzu reálně provést a výsledky byly vzhledem k okolnostem důvěryhodné.

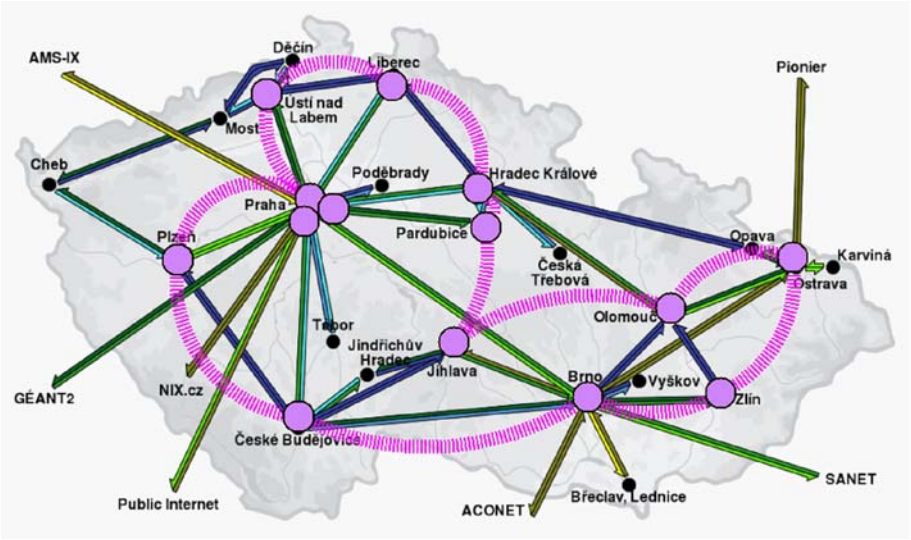
2. Způsob ukládání, uchování a vyhledávání dat musí být takový, aby při zpětném vyhledávání bylo možné očekávat výsledky s přijatelnou dobou odezvy, a to i v případě komplexních vyhledávacích podmínek a rozsahu vyhledávání v řádech desítek hodin. To vše při paralelním bezvýpadkovém plnění úložiště aktuálně přicházejícími novými provozními záznamy.

Systémů na zpracování IP provozních záznamů a souvisejících podpůrných nástrojů a „udělátek“ je k dispozici celá řada, a to jak v komerční, tak ve volně šiřitelné podobě. Vždy je nutné nejprve pečlivě zvažovat cíle a opakovaně analyzovat vlastní požadavky – množství a typy očekávaných výstupů a architekturu celého schématu sledování IP provozu a potom se teprve rozhodovat zda zvolit hotový systém, nebo využít množinu polotovarů a nad nimi vybudovat vlastní nadstavbu, či vybudovat o základů systém vlastní ušitý na míru vlastním požadavkům.

Nástroj pro plošné a souvislé zpracování provozních informací na bázi IP toků v síti CESNET2

Síť CESNET2 je hybridní multi-gigabitová velká infrastruktura jejíž páteřní IP síť má architekturu IP/MPLS jádra s hranicemi přibližně vymezenými jednotlivými gigabitovými místy přítomnosti v rámci celé ČR. Zdroji IP provozních záznamů jsou páteřní směrovače tvořící pomyslný vnější obvod IP/MPLS páteřní sítě. Topologické rozmístění zdrojů je patrné z obrázku 5.

Pro plošné, souvislé zpracování provozních záznamu používáme systém FTAS (Flow-based Traffic Analysis System), který je sdružením CESNET vyvíjen v rámci řešení výzkumného záměru „Optická síť národního výzkumu a její nové aplikace“.



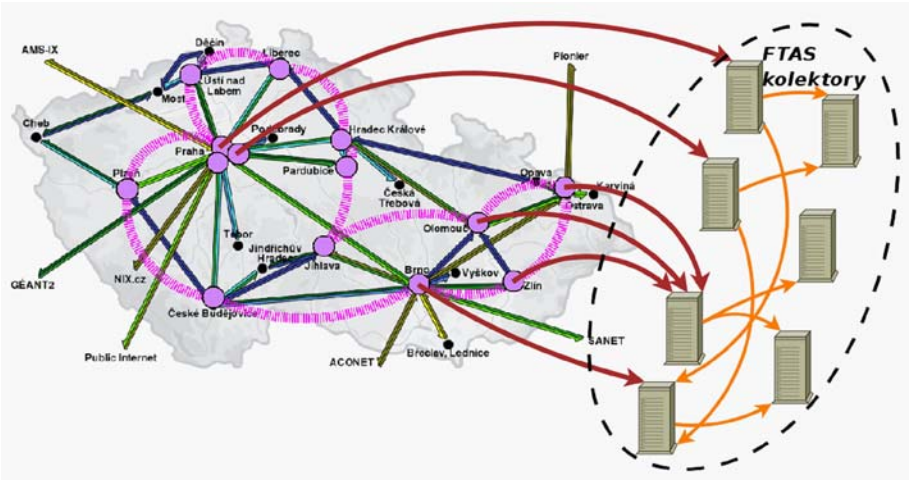
Obr. 5 Rozsah sběru provozních záznamů v IP/MPLS páteři sítě CESNET2

Systém je modulární a umožňuje distribuované zpracování provozních záznamů. Základním prvkem je víceúčelový FTAS kolektor, kterých může být v rámci jedné instance systému teoreticky neomezené množství. Všechny FTAS kolektory jedné instance systému se opírají o centrální konfiguraci (libovolně umístěnou). FTAS kolektor se skládá ze dvou logických celků – úložiště (lokálního nebo vzdáleného) a sady 1-M (na základě aktuální konfigurace) víceúčelových „procesorů provozních záznamů“, což jsou de-facto samostatné procesy pro konkrétní způsob zpracování provozních záznamů v rámci hostitelského stroje. Úložiště je zpravidla (na základě konfigurace) sdílené procesory provozních záznamů v rámci jednoho kolektoru. Systém je z hlediska výkonu škálovatelný až do nejnižší úrovně 1 : 1 (jeden zdroj provozních záznamů na jeden FTAS kolektor). Systém podporuje všechny aktuální exportní formáty provozních záznamů. IPv6 je podporováno jak na úrovni importu provozních záznamů, tak v rámci interní redistribuce provozních záznamů (v tomto případě řízeně na základě konfigurace).

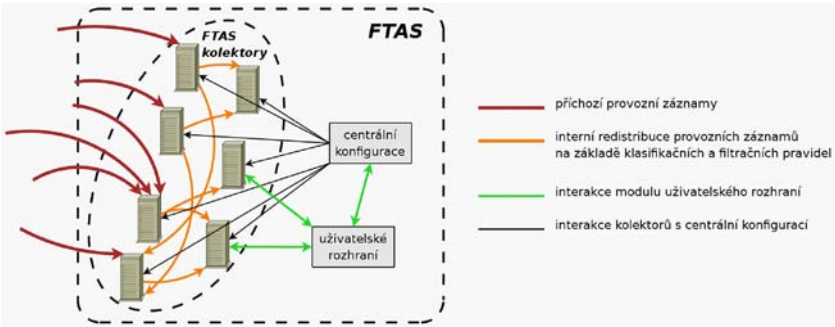
Administraci i běžnou práci se systémem zajišťují „webem“ distribuovaná uživatelská rozhraní. Systém je provozován na běžně dostupném HW serverového typu vyšší třídy (vyšší nároky na I/O průchodnost) pod OS Linux. Úložiště je realizováno pomocí MySQL, systém FTAS je naprogramován v jazyce Perl.

Aktuálně běžící instance systému FTAS pro IP/MPLS páteř sítě CESNET2 sestává z 7 serverů. Provozní záznamy jsou distribuovány v poměru 1 : 1 až

6 : 1 (z hlediska počtu zdrojů na jeden FTAS kolektor) v rámci systému pak dochází k dalším možným redistribucím na základě filtračních a klasifikačních podmínek – principiální schéma je znázorněno na obrázku 6. Na obrázku 7 jsou pak znázorněny logické vazby jednotlivých komponent systému.



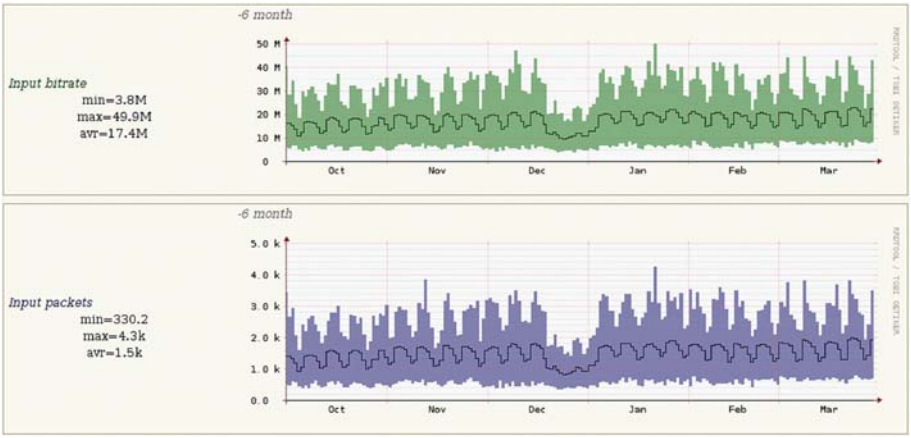
Obr. 6 Schéma exportu provozních záznamů z IP/MPLS páteře sítě CESNET2 do systému FTAS



Obr. 7 Logické vazby jednotlivých komponent FTAS systému

Objemy provozních záznamů v IP/MPLS páteři sítě CESNET2

Jak již bylo uvedeno výše, je pro účely řešení analýzy provozu z důvodu bezpečnostních hlášení optimální mít k dispozici maximálně věrný obraz provozu sítě. To nemusí být vždy možné v souvislosti s generovaným množstvím provozních informací a vzhledem k maximálnímu výkonu celého řetězce pro zpracování. Na obrázku 8 je pro představu uvedena dlouhodobá (půl roku do konce března 2009) agregovaná statistika statistika celkového toku provozních záznamů z primárních zdrojů IP/MPLS páteře sítě CESNET2 do systému FTAS.

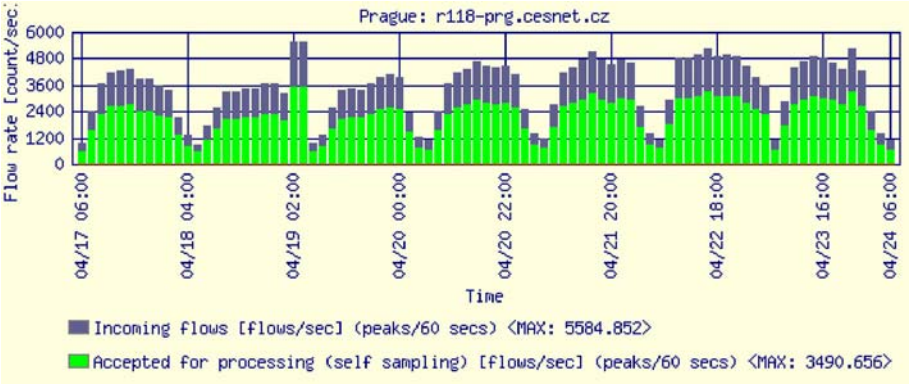


Obr. 8 Agregovaný průběh objemu toku provozních záznamů z IP/MPLS páteře sítěCESNET2 do systému FTAS

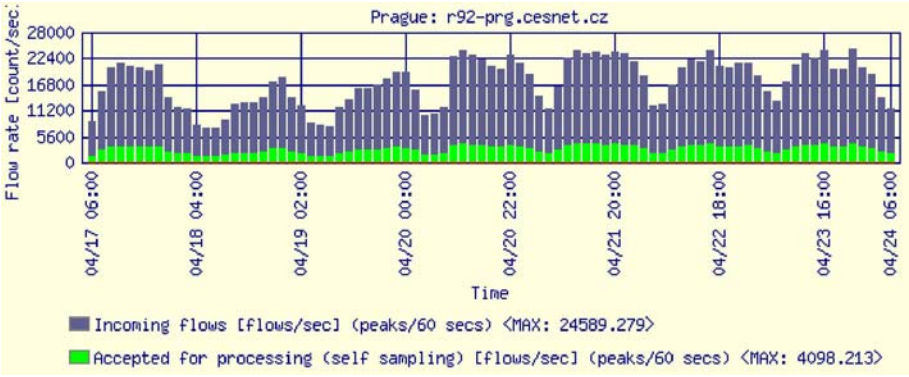
Při průměrné délce provozního záznamu okolo 50–60 B v závislosti na exportní verzi (a včetně podílu záznamu na délce hlaviček exportních formátů) reprezentuje uvedený průběh v průměru 36 000 provozních záznamů ke zpracování za vteřinu a ve špičkách přes 100 000 záznamů za vteřinu. Z hlediska např. měsíčního objemu reprezentuje tento tok přes 5,6 TB provozních dat v základní podobě (ve stejné datové reprezentaci jako v exportním formátu).

Tento celý objem provozních záznamů v síti CESNET2 nezpracováváme. Naší snahou je udržet příznivý poměr cena/výkon při zachování snesitelné ztráty vypovídací hodnoty a systém provozujeme na architektonicky standardním HW. Aktuálně nastavené hodnoty vzorkování na úrovni exportovaných záznamů v rozsahu 1 : 1,6–1 : 7 (a v případě jednoho zdroje 1 : 50 na paketové úrovni) snižují průměrné množství provozních záznamů z jednotlivých zdrojů, které je dále nutné zpracovávat na úroveň cca 3 000/s–4 000/s na jeden FTAS kolektor. Toto

snížení vypovídací hodnoty dat z jednotlivých zdrojů je díky architektuře sítě kompenzováno tím, že většina provozu je přenášena přes více než jeden zdroj provozních záznamů. Příklady průběhů příchozích i redukovaných objemů příchozích provozních záznamů jsou na obrázku 9, kde se jedná o zdroj i s povinným vzorkováním na úrovni paketů (1 : 50) a na obrázku 10, kde se jedná o zdroj bez paketového vzorkování.



Obr. 9 Množství emitovaných a zpracovávaných provozních záznamů z jednoho zdroje v IP/MPLS páteři sítěCESNET2 – zdroj se vzorkováním na paketové úrovni



Obr. 10 Množství emitovaných a zpracovávaných provozních záznamů z jednoho zdroje v IP/MPLS páteři sítěCESNET2 – zdroj bez vzorkování na paketové úrovni

V souhrnu se potom v IP/MPLS páteři sítě CESNET2 jedná v průměru o cca 17 000 dále zpracovávaných provozních záznamů/s. Součástí dalšího zpracování systémem FTAS jsou i sekundární interní redistribuce záznamů – ty reprezentují aktuálně dalších cca 12 000 záznamů/s ke zpracování.

Doba uchovávání provozních záznamů z primárních zdrojů, což jsou ta data, která nám umožňují provádět provozní analýzy na základě bezpečnostních hlášení, se aktuálně pohybuje okolo 60 dnů.

Závěr

Plošné a souvislé sledování IP provozu je bezesporu významnou přidanou hodnotou správy sítě, a to nejen pro oblast řešení bezpečnostních incidentů. V síti CESNET2 bylo například za posledních 12 měsíců prostřednictvím systému FTAS provedeno okolo 10 000 vyhledávání v provozních informací, což je vzhledem k velmi omezenému spektru uživatelů (správci páteřní sítě a bezpečnostní týmy) relativně vysoké číslo (v průměru více něž 800 za měsíc). Na druhou stranu se jedná o poměrně komplikovanou problematiku z hlediska přijatelného poměru nároků na zdroje a výsledného efektu. Na základě vlastní zkušenosti si troufám tvrdit, že v této oblasti je třeba vždy hledat filosofii sběru a zpracování provozních informací v závislosti na konkrétních podmínkách (typ sítě, architektura sítě, politika směrování, aktivní prvky sítě, struktura a typ uživatelů apod.) a teprve potom začít budovat řešení ušité na míru – a jak již bylo řečeno - bez ohledu na to, zda se bude jednat o hotový systém nebo vlastní systém. Výsledným efektem je potom obraz toho, co se v síti reálně děje, a to jak v souhrnu, tak i na úrovni jednotlivých přenosů dat mezi konkrétními uzly sítě. Takové informace mají z pohledu správy sítě a bezpečnostních týmů rozhodně nezanedbatelnou hodnotu.

Data Leak Prevention/Data Loss Protection

Martin Zich

E-mail: martin.zich@hp.com

Abstrakt

Problém úniku citlivých dat ze soukromých a firemních sítí je stále aktuálnější. Zvláště dnes v období finanční krize se ztráta konkurenční výhody v důsledku úniku tajných informací může rovnat zániku poškozené firmy. Systémy souhrnně označované jako DLP (Data Leak Protection) se snaží takovým ztrátám co možná nejúčinněji zabránit. První část článku obsahuje několik motivačních příkladů, které zřetelně ilustrují stále větší potřebu a aktuálnost zavedení DLP řešení v korporátních sítích. Další části se pak zaměřují na historický vývoj a konečně popis konkrétních vlastností a postupů, které využívají především top hráči na tomto mladém a perspektivním trhu. Tento text se neupíná ke konkrétním proprietátním řešením, ale představuje produkty DLP jako celek. Poskytuje tak obecný přehled napříč všemi dostupnými implementacemi a dobře tak poslouží při rozhodování o zavedení konkrétního produktu ne podle maketingové kampaně, ale na základě konkrétních služeb, které je DLP jako technologie schopno nabídnout.

1 Různě to samé

Systémy DLP jsou historicky označovány mnoha různými jmény. Jako nejpoužívanější lze označit Data Leak Protection/Prevention a Data Loss Protection/Prevention, ze kterých vznikla i samotná zkratka DLP. Dalšími již méně používanými názvy jsou pak Information Loss Prevention/Protection, Information Leak Prevention/Protection, Extrusion Prevention, Content Monitoring and Filtering, Content Monitoring and Protection. Nejjednodušší a nejpoužívanější je však stále varianta samotné zkratky DLP. Již z existence velkého množství mnohdy zcela odlišných názvů lze odvodit, že není přesně definováno co ještě je a co už není DLP systém. Různí výrobci chápou obsah tohoto označení odlišně. Existuje však jádro funkcionalit, které jsou u většiny produktů velmi podobné až shodné. Rozbor především tohoto jádra je předmětem následujíciho textu.

2 Co je to DLP

V nepříliš vzdálené historii bylo nutné ochránit počítačovou síť výhradně před útoky typu virové nákazy, zanesení spyware nebo spamu. Snahou bylo klást důraz pouze na řešení představující ochranu před nebezpečnými útoky směřujícími z prostředí Internetu nebo jiného externího zdroje. S jistou mírou abstrakce lze říci, že hlavním úkolem bylo řešit problémy spojené s pohybem dat zvenku směrem dovnitř navíc s důrazem na ochranu infrastruktury. Novým přístupem je doplnění těchto principů další úrovní kontroly informací podle obsahu jak uvnitř sítě, tak při přenosu směrem ven. Mezi hlavní potřeby firem dnes patří monitoring obsahu dat obsahujících citlivé informace. Je nutné znát pohyb takových dat a blokovat či jinak zasáhnout proti případnému úniku. V konečných součtech totiž takové události vedou k obrovské porci ztracených finančních prostředků.

Předcházení nebo zmírnění finančních ztrát v souvislosti s úniky citlivých informací si kladou za cíl dnešní moderní DLP řešení. Hned na úvod je však třeba podotknout, že DLP není primárně určeno k souboji s hackery, kteří se snaží za každou cenu vydolovat konkrétní data. Na DLP se často snáší velká vlna kritiky za to, že téměř vždy jde obejít, a že tudíž nedokáže ochránit data proti jejich úniku i když to svým názvem jasně deklaruje. Ve většině případů plynou argumenty odpůrců z nepochopení toho, k čemu má DLP primárně sloužit. Jde totiž o systém, který pomůže naučit zaměstance zacházet s citlivými informacemi, předejde nechtěným únikům v důsledku chyby pracovníka, zabrání automatickým nepozorovaným ztrátám informací z důvodu činnosti škodlivého software nebo ztíží situaci případným úmyslným narušitelům natolik, že o data ztratí zájem. DLP pomůže přeměnit počáteční chaos v tom kde a jaká citlivá data ve firmě jsou a také kdo a jak s nimi nakládá. Poskytne odpovědi na kritické otázky uvedené na obr. 1.

Pokud dojde k úniku, je snadné dohledat příslušnou cestu a zodpovědné osoby. To se hodí v těch závažnějších případech, kdy celá věc dospěje až k soudnímu řízení.

Velké množství nejrůznějších studií ukázalo, že k nejzávažnějším a především nejčastějším ztrátám s následným obrovským finančním dopadem dochází z důvodu nedbalosi nebo chyby zaměstance. To znamená, že většinu tvoří úniky neúmyslné. Tuto skutečnost právě zohledňuje i DLP. K dispozici je však i řada prostředků jak zabránit případným úmyslným pokusům připravit firmu o její informace. Je jasné, že žádný softwarový produkt nikdy nezabrání ve fyzickém provedení krádeže. Pokud se dotyčný vloupá do objektu firmy a vyfotí si obrazovky s citlivými údaji digitálním fotoaparátem, DLP nám rozhodně žádné upozornění nepošle. Nicméně to, že se data na daném počítači nacházejí, že byla

Obr. 1 Otázky týkající se citlivých dat, na které DLP poskytne odpovědi

otevřena, kdy k tomu došlo, zda proběhl pokus o jejich transport atd. se dozvíme zcela bezpečně. Zloděj si je také nebude moci odnést například v kopii na USB disku nebo vytištěné, protože taková akce bude zablokována nebo budou data při přenosu zašifrována. Možností je v tomto směru velká spousta. DLP tedy nesmí být bráno jako řešení na 100% ochranu dat za jakýchkoliv okolností. Je vždy součástí komplexnějšího systému bezpečnosti té které firmy. Lze tedy říci, že DLP je systémem, který se snaží poskytovat cesty jak případné úmyslné narušitele přinejmenším odradit nebo jim situaci ztížit až do té míry, že pro ně případná krádež přestane být výhodná. Nedbalostní záležitosti však eliminuje takřka úplně a v tom je nutné spatřovat jeho hlavní přínos.

3 Proč je DLP systém tak potřebný

Podle studií provedených v souvislosti s únikem informací bylo zjištěno, že ke ztrátám dochází především čtyřmi hlavními cestami. Konkrétně jde o úniky v důsledku krádeže přenosného zařízení (laptop, handheld, PDA, apod.), možnosti neomezeného kopírování informací na USB klíče a vypalování na CD/DVD, přenosu prostřednictvím emailu a konečně volného tisku dokumentů. Jako zástupci v minulosti zaznamenaných incidentů byly vybrány následující příklady.

3.1 Ztráta CD s osobními údaji

K jednomu z velkých úniků došlo v nedávné době ve Velké Británii. CD s údaji o sedmistech tisících britských domácností, které pobírají přídavky na děti se

ztratilo neznámo kam. Johnathan Bamford z Information Comissioner's Office, což je britská varianta našeho Úřadu na ochranu osobních údajů, se nechal slyšet, že se jedná o největší pohromu v oblasti bezpečnosti, kterou kdy Británie zažila.

DLP: Pokud by bylo pomocí funkčního DLP stanoveno, že daná data nesmí být na CD vůbec kopírována, k úniku by touto cestou rozhodně nedošlo. Možné by také bylo data při kopírování šifrovat.

3.2 Krádež pevného disku

Podle informací zveřejněných 10. října 2008 v [1] došlo na britském ministerstvu obrany ke ztrátě pevného disku z počítače spolu s osobními údaji o přibližně 100 000 příslušnících armády a asi 600 000 uchazečích.

DLP: Je otázkou zda ministerstvo vůbec vědělo, že skladuje takové množství citlivých údajů na jednom místě. Systém DLP by na to zcela jistě upozornil a minimálně zmírnil případnou ztrátu již předem.

3.3 Nechtěné zveřejnění osobních údajů

I norské ministerstvo financí mělo podle zprávy ze 17. září uvedené v [2] co vysvětlovat. Zveřejnilo totiž pravidelnou zprávu o výši ročních příjmů a daňových platbách ekonomicky aktivních Norů. Tyto údaje jsou v zemi obecně přístupné. Do systému se ale vloudila chybička a úřad s těmito údaji zveřejnil i rodná čísla všech těchto občanů.

DLP: DLP jako kontrolní vrstva pohybu citlivých údajů (například rodných čísel) by na chybu upozornilo. Následné zveřejnění na webových stránkách by nemohlo proběhnout, protože by bylo zablokováno.

4 Jaké informace jsou tak citlivé?

Pokud se stále hovoří o ochraně citlivých informací, je nutné uvést nějaké konkrétní příklady. Která to vlastně jsou ta data, která nesmí nikam nekontrolovaně odejít. Odpověď je velmi jednoduchá. Pro podnikatele to mohou být data finančního rázu, všechny uzavřené smlouvy a kontrakty a také osobní údaje zaměstnanců nebo klientů. Mezi další patří zdrojové kódy, data konkurenčních výhod, specifikace stávajících a plánovanách projektů, data vývojových center apod. Tento výčet samozřejmě není úplný, nicméně pokrývá kategorie těch nejcennějších informací, které se ve firemních sítích běžně vyskytují.

5 Historicky jednoúčelově vs. Moderně komplexně

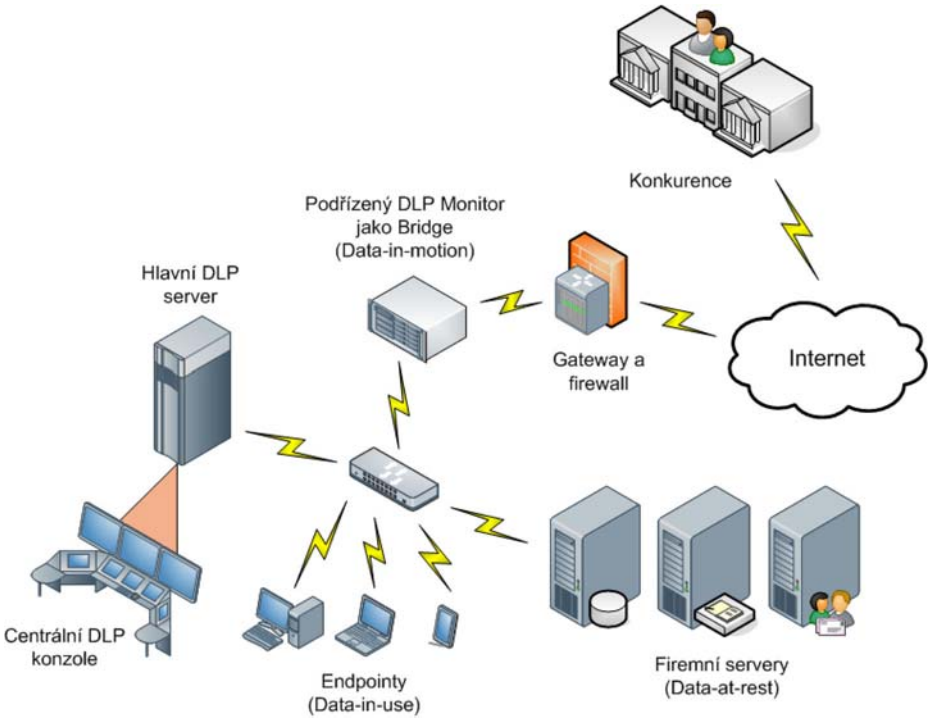
Pokud budeme hledat nějaké kořeny nebo historické předky dnešních DLP, určitě je nalezneme v aplikacích, které se typicky zaměřovaly na jeden konkrétní problém a ten se snažily řešit. Mohlo se jednat o software umístěný na vstupním prvku sítě porovnávající provoz s vlastní databází vzorků nebo aplikaci na cílové stanici, která hlídala kopírování dat na vyměnitelná zařízení. V síti to dále byl třeba i inteligentní hardware zachytávající pakety a provádějící jejich následnou analýzu. Rozdíl mezi těmito aplikacemi a moderním DLP je především v tom, že nová řešení největších softwarových hráčů nabízejí kombinaci všech služeb v jednom boxu. Jasná je snaha poskytnou vše pěkně *all-in-one*, což nabízí jako jednu ze svých výhod centrální management, přehledné logování a reporting. Oba přístupy přetrvaly i do dnešní doby nicméně je jasné, že pro velké implementace se jeví jako vhodnější komplexní produkt s efektivní správou místo velkého počtu autonomních nástrojů. Menší firmy však nemusí využít všechny nabízené parametry, a tak dají přednost něčemu jednoúčelovému. Je však dobré dát velký pozor, aby současné rozhodnutí nebylo příliš krátkozraké a nezpůsobilo víc škody než užitku při dalším rozšiřování.

6 Základní architektura

Základní architekturou je klient-server. Centrální server, na kterém běží DLP, komunikuje s klienty umístěnými na koncových stanicích. Někdy bývá serverů více, když každý podlehá tomu hlavnímu a řeší svojí konkrétní technologickou oblast v rámci DLP. Klienti na stanicích jsou označováni jako agenti. Umožňují nepřetržité fungování systému DLP i když je počítač odpojen od sítě. Agenti si potřebné informace sami synchronizují s centrálním serverem v době, kdy se nacházejí v jeho dosahu. Jejich instalace probíhá u většiny produktů automaticky a pro uživatele zcela neviditelně. Klientem v představené architektuře může kromě pracovní stanice i celá databáze nebo fileserver. Jednotlivá místa podléhající správě DLP se obvykle definují přímo v centrální konzoli.

7 Tři pohledy na data v DLP

Z pohledu DLP se vžilo dnes již téměř standardní označení druhů dat v korporátní síti. Data jsou pojmenována podle toho, v jakém stavu se právě nacházejí a to konkrétně Data-at-rest (Data uložená), Data-in-motion (Data v pohybu), Data-in-use (Data používaná).



Obr. 2 Architektura jednoduchého DLP řešení

7.1 Data-at-rest

Jak již samotný název napovídá, jedná se o data, která někde odpočívají. Jsou uložená a čekají na to, že se dostanou do pohybu nebo budou nějak použita. Mohou se nacházet na pracovních stanicích uživatelů, zálohovacích systémech, vzdálených úložištích (SAN), externích médiích, mail-serverech, file-serverech a v různých druzích databází. Stojí za zmínku, že se nejedná o data, která jsou uložena v operační paměti. Podle [5] používá DLP k prozkoumávání takto uložených informací obecně tři různé způsoby.

- **Vzdálený scan:** Centrální server se v tomto případě připojí ke vzdálenému úložišti a provede vyhledání citlivých dat.

- **Scan instalovaného agenta:** Agent je přímo nainstalován na cílový systém. Výhodou tohoto řešení je využití výpočetních prostředků zkoumaného uzlu sítě bez zbytečného zatížení centrálního serveru.

- **Scan agenta v paměti:** Jedná se o podobný přístup jako v předchozím případě s tím rozdílem, že agent je po celou dobu své životnosti uchováván pouze v operační paměti cíle.

Moderní řešení používají ve většině případů kombinaci těchto metod například podle typu cílového systému. Provedením scanu získáme informace, kde se citlivá data nalézají. Jedná se vlastně o provedení něčeho jako inventury v obyčejném obchodě. V mnohých případech jsou administrátoři přinejmenším překvapeni, kde všude se podařilo citlivé informace dohledat.

7.2 Data-in-motion

Pokud se odpočívající data dostanou do pohybu, stávají se z nich Data-in-motion. Může se jednat o jejich přesun nebo kopírování mezi pracovními stanicemi nebo file-servery, přenos pomocí emailu nebo obecný přenost do/z jiných sítí. Patří sem také cesty mezi databází a stanicí s ní komunikující.

Typickou komponentou DLP starající se on tento segment dat je Network Monitor (dále jen NM). Řešena je především komunikace mezi firemní sítí a okolním prostředím a tak se zmíněný NM nachází blízko hlavní gateway. V síti se obecně může nalézat více NM, které si stahují politiky a posílají svoje reporty na centrální DLP server. Většina řešení je pouze softwarová. Přesto stále více produktů obsahuje i vlastní vyladěný hardware. Monitor umí zachytávat všechny pakety a analyzovat jejich obsah. Vše probíhá realtime. Každého jistě napadne, zda dochází k degradaci rychlosti síťové komunikace po zapnutí takového prvku. Typický objem přenesených dat v malé firmě je v řádech desítek MB za vteřinu, středně velké firmě dosahuje až stovek MB a pokud se jedná o velkou společnost tak může přenos pohybovat na úrovni GB. Dostupné produkty nemají i s takovým objemem dat problémy a podle jejich výrobců Vám síť nezpomalí a jejich zapnutí takřka nezaregistrujete. NM může podle konkrétního řešení pracovat v různých módech. Mezi nejčastější patří:

- **Most:** V tomto módu jsou pakety analyzovány po jednom a přeposílány. Pokud je však zjištěna potřeba blokovat data z obsahu dalších později přichozích paketů, může již být pozdě.
- **Proxy:** Proxy tvoří frontu komunikace a analyzuje větší celky dat. S výhodou mohou být použity existující proxy (HTTP, FTP).

Jednou z dalších, často se vyskytujících komponent je specializovaná část řešící emailovou komunikaci. Většinou jde o nějakého mail agenta, který zkoumá obsah posílaných zpráv. Emailová komunikace je specifická tím, že ji lze popsat jako *store and forward*. Mail agent může rozhodnout o emailu jako celku a tak ho zašifrovat, odložit do karantény, klonovat na jiné místo apod.

7.3 Data-in-use

Jako Data-in-use jsou označována všechna data, se kterým je nakládáno v rámci koncové pracovní stanice. Mohou být čtena nebo upravována, přenášena na vyměnitelná zařízení (CD/DVD, Ipody, USB klíče atd.), uploadována pomocí HTTP POST popřípadě kopírována na clipboard. Podobných akcí je velká spousta, nicméně lze je obecně rozdělit do tří skupin.

- **Síťové přenosy:** Sem patří monitoring veškeré komunikace přes síťová rozhraní stanice. Vzhledem k tomu, že nastavené politiky jsou vynucovány instalovaným agentem, je samozřejmě možné pracovat dočasně i v jiné než firemní síti nebo offline.
- **Lokální akce:** Jde především o vkládání, kopírování a modifikace místně uložených citlivých informací.
- **Interakce s periferními zařízeními:** Lze definovat co se s daty stane pokud jsou přemistována mezi stanicí a nějakou vyměnitelnou pamětí. Lze nastavit blokování, šifrování, klonování na jiná místa, elektronické podepisování atd.

8 Hlavní pilíř DLP

Hlavním pilířem DLP je analýza obsahu dat. Nepoužívá se samozřejmě jen jediná technika, ale množství rozdílných algoritmů, které jsou více či méně vhodné ke konkrétnímu účelu. Většina produktů podporuje velkou škálu typů souborů jako jsou dokumenty, obrázky, zdrojové kódy, CAD soubory apod. To vše v různých jazykových mutacích včetně podpory těch asijských. Analyzovat lze dokonce i komprimovaná nebo šifrovaná data. K dešifrování musí mít DLP k dispozici příslušné soukromé klíče.

Pro analýzu obsahu se používá množství nejrůznější algoritmů. Zde jsou uvedeny ty základní nebo ty, které se vyskytují téměř ve všech DLP produktech. Více příkladů lze nalézt v [5] nebo [6].

- **Metoda vzorků:** Tato metoda je základem každého DLP. Využívá databázi vzorků textu (regulární výrazy) vzhledem ke které se prohledává. Metoda těží z rychlé konfigurace a je zvykem, že první množinu vzorků již produkt obsahuje při jeho dodání výrobcem. Problémy jsou spojeny s vysokou mírou tzv. *false positives*. Regulární výraz může pokrývat i řetězce, které za nositele citlivé informace původně nebyly považovány. Vyhledávací engine to však nedokáže rozlišit. Pro nestrukturovaná data nelze tento přístup použít vůbec.

- **Metoda otisků konkrétních dat:** V rámci tohoto přístupu je třeba definovat konkrétní informace, které mají být řešeny jako citlivé. Z dat je vytvořen otisk, který je následně používán pro srovnávání. Uvedený postup se používá především k prohledávání databází. *False positives* jsou zde zcela eliminovány.

- **Metoda přesné shody souborů:** Na počátku je potřeba pořídit hash souboru, který bude následně chráněn. Tento způsob se hodí ke správě mediálních dat, kde selhává textová analýza. O *false positives* nelze téměř hovořit a výhodou může být i podpora jakéhokoliv formátu. Každého však napadne, že při jakékoliv změně je hash jiný a soubor nalezen nebude. Proto tento přístup není vhodný pro ochranu snadno editovatelných textových souborů.

- **Metoda shody částí dokumentů:** Do systému je zaregistrován dokument, který obsahuje citlivé informace. DLP pak dokáže sledovat i nakládání s jeho částmi. To znamená, že pokud někdo vezme jen kus z tohoto zdroje, pozornosti DLP to neunikne. Z pohledu implementace se většinou jedná o použití cyklického hashování v kombinaci s počítáním pravděpodobnosti jednotlivých slov. Vhodným použitím této metody se zdá být ochrana různých zdrojových kódů, CAD souborů i nestrukturovaných dat. Ze zkušeností jsou hodnoty *false positives* v tomto případě poměrně nízké. Nicméně již proti použití opravdu základního „šifrování“ například rotací abecedy o 1 pozici, nemá tato metoda žádnou šanci.

- **Statistická analýza:** Metoda statistické analýzy je samozřejmě založena na množství výskytu různých vzorových slov a výrazů. Pokud je špatně navržena může potenciálního uživatele zahltit hromadou *false positives*. Hodí se především pro běžně čitelné dokumenty zatímco u binárních dat její přínos lze hledat jen těžko.

- **Lexikon:** Některé produkty mají prostředky pro detekci nestandardního chování uživatele. To znamená, že lze zjistit, zda si například nehledá nové zaměstnání, neprovozuje nějaký sexual harassment apod. Protože se mnohdy jedná o velmi vágně definovaná pravidla pomocí nejrůznějších kombinací předchozích technik, může míra *false positives* také poměrně jednoduše přerůst únosnou mez.

9 Nasazení

Problém nasazení systému DLP se nezdá být triviální. Opak je však pravdou, pokud se zvolí správný postup. Ta nejlepší řešení obsahují spoustu automatizmů, které celý proces nesmírně ulehčí a především urychlí. Kroky k úspěšnému

nastartování ochrany citlivých informací se mohou u různých produktů poměrně lišit. Některé mají již přednastaveny nejrůznější *templaty* jak to již bývá u podobných řešení zvykem. U jiných je potřeba stanovit od nuly, která jsou ta citlivá data o něž se nechceme s konkurencí dělit. Teoretický a zjednodušený scénář nasazení popisuje následující odstavec. Popis podrobnějšího postupu poskytuje oddíl 9.2 tohoto dokumentu.

Obr. 3 Úvodní nasazení DLP řešení

9.1 Obecný postup zavádění DLP řešení

1. **Definice citlivých informací:** Na počátku celého procesu je třeba stanovit, které informace budeme chtít ochránit. Typicky se sejdou zástupci všech hlavních oddělení ve firmě jako je HR, IT, právní oddělení atd. a dohodnou se na úvodní malé množině dat, kterou bude řešit DLP. Jak již bylo zmíněno, některé nástroje umožnují tento proces urychlit nebo dokonce přeskočit, protože již obsahují typické množiny citlivých dat přímo od výrobce (např. formáty kreditních karet, lékařských záznamů, platových výměrů atd.).

2. **Úvodní tvorba politik a kategorií dat:** Informace, které bude potřeba chránit jsou rozděleny do kategorií, pro které jsou stanoveny jednotlivé politiky a konkrétní pravidla.

3. **Ladění pravidel a zpětná vazba:** Na základě dosahovaných výsledků je potřeba nastavené politiky dále ladit. Provádí se úpravy vzorků, volba správných algoritmů pro jednotlivé typy dat atd. Stojí za zmínku, že všechny produkty poskytují možnost tzv. pomalého náběhu. To znamená, že v prvním kroku jsou politiky v módu upozorňování a stanovená pravidla přímo nevynucují. V tomto momentě je potřeba analyzovat, jaké by byly dopady při případném zapnutí nekompromisního vynucení pravidel.

Je vhodné žádat zpětnou vazbu od koncových uživatelů a skutečně sledovat případné změny a omezení jejich práce, aby se předešlo pozdější nežádoucí degradaci produktivity.

4. **Zapnutí vynucení politik:** Teprve po odladění je přistoupeno k plnému běhu a tudíž blokování úniku citlivých dat.

9.2 Podrobnější postup přidávání nových množin informací

Pro přidávání dalších informací, které je potřeba chránit, je vhodné zavést nový proces. Jednotlivé kroky jsou velmi pěkně ilustrovány v [3]. Dodržení nastíněného postupu může pomoci předejít velkému množství komplikací. Přidání nové množiny tedy prochází těmito logickými kroky:

1. Oddělení firmy požádá o ochranu nějakého obsahu DLP administrátory.
2. Na schůzce mezi těmito skupinami jsou dohodnuty cíle a požadavky nové ochrany.
3. DLP administrátoři kontaktují právní oddělení firmy, aby konzultovali možná rizika a omezení.
4. Následně je implementována první verze nové politiky.
5. Politika není hned od počátku tvrdě vynucována, ale nastavena pouze k upozornění uživatele o provádění nepovolených operací. V tomto bodě dochází také k ladění jednotlivých pravidel.
6. DLP administrátoři definují příslušné workflow pro zacházení s informacemi s přihlédnutím k nové politice a informují o něm koncové uživatele.
7. Na další schůzce zhodnotí DLP administrátoři spolu se zástupci žádájícího oddělení firmy, zda byly splněny všechny prvotní požadavky.
8. V dalším testování je již vynucováno nové workflow, avšak vynucení politik je stále nahrazeno pouhým upozorňováním.
9. Pokud je nová politika po nějaké době shledána jako stabilní, je možné postoupit dále. V případě, že se objevují jakékoliv problémy, je nutný návrat k některému z předchozích bodů.
10. Dodržení nových pravidel je nakonec vynucováno v plném rozsahu.

Po provedení těchto kroků je jasné CO, KDE a JAK je chráněno. Postup platný pro přidávání nových pravidel a politik platí samozřejmě i pro první zavedení systému. Vždy je potřeba klást důraz na testování, zhodnocení funkčnosti a také nezanedbat poučení zaměstatnců před tvrdým vynucením dodržování nových pravidel.

9.3 Řešení incidentů

Po zavedení je potřeba určitým způsobem umět reagovat na nastalé incidenty. Je vhodné i pro tuto situaci stanovit konkrétní postup. Článek uvedený v [4] poskytuje asi nejlepší možný přístup.

1. Je detekováno porušení pravidel pro zacházení s citlivými informacemi a zařazeno do fronty některého ze zpracovatelů.

2. Zpracovatel potvrdí incident a přiřadí úroveň závažnosti.

3. Pokud je zapotřebí provést nějakou další akci, je incident eskalován a otevřeno jeho vyšetřování.

4. Je kontaktováno oddělení, které vlastní příslušné citlivé informace a upozorněno na zaznamenání incidentu.

5. Incident je zhodnocen oběma stranami a jsou provedeny všechny potřebné kroky k okamžité ochraně informací.

6. Uživatel je upozorněn, že narušil nastavenou politiku (např. emailem nebo jiným způsobem).

7. Pokud je to nutné, je kontaktován i uživatelův manažer, popřípadě HR oddělení.

8. Uživatel je poučen o správném postupu v podobných situacích. V závažnějších případech jsou provedeny další kroky v rámci příslušných oddělení firmy (HR, právní apod.).

9. Incident je uzavřen.

10 Vývoj na trhu s DLP

Trh s DLP produkty se začíná značně rozrůstat. Nejznámější značky do něj vstupují především formou akvizic malých firem a růst konkurence se zdá být raketový. Potenciální zákazníci do nedávné doby razili taktiku vyčkávání a mapování trhu. Jedním z nejčastějších požadavků byla podpora šifrování nebo blokace dat při jejich přenosu na periferní zařízení. V dnešních dnech, kdy se díky finanční krizi stávají informace ještě cennějším zbožím, přistupuje mnoho firem k samotné realizaci. V roce 2009 je navzdory nebo možná díky celosvětové recesi očekáván velký rozmach DLP a obecně všech technologií na ochranu konkurenčně i marketingově zajímavých dat.

11 Konkrétní implemenatce

Na závěr je uveden seznam některých konkrétních dnes dostupných produktů. Mezi ty, které si zaslouží větší pozornost patří: Symantec – Data Loss Prevention (dříve Vontu), CA – Orchestia DLP, McAfee – Network Data Loss Prevention (dříve Reconnex iGuard), Websense – Content Protection Suite, Verdasys – Digital Guardian, InfoWatch – Traffic Monitor, Code Green Networks – TrueDLP Solution, RSA – Data Loss Prevention (DLP) Suite, Trend Micro – LeakProof, Vericept – Data Loss Prevention Solution, Workshare – Protect a další.

Reference

[1] http://www.idnes.cz Britské ministerstvo obrany ztratilo údaje o polovině svých vojáků, 10. 10. 2008.

[2] http://www.idnes.cz Norská vláda omylem zveřejnila rodná čísla občanů, 17. 9. 2008.

[3] http://securosis.com/ Best Practices For DLP Content Discovery: Part 4, rmogull, 29. 8. 2008.

[4] http://securosis.com/ Best Practices For DLP Content Discovery: Part 5, rmogull, 29. 8. 2008.

[5] http://securosis.com/ Understanding and Selecting a Data Loss Prevention Solution, Securosis & Sans Institute, 2008.

[6] http://www.demosondemand.com/DemoStage2/index_new.asp?sessID=WEBS006&promotion_id=2143 Websense Data Security Suite, Websense 2008.

[7] Data Leak Prevention, The Forrester Wave 2008.

[8] How to avoid DLP implementation pitfalls, Rich Mogull, 8. 21. 2008.

[9] The ROI of Data Loss Prevention (DLP), Websense 2008.

Jak se smaží zásobník

Radoslav Bodó

E-mail: bodik@civ.zcu.cz

Abstrakt

Hitem současných let je a bude web, webové aplikace, SOA, XML, ... Tyto nové, úžasné technologie však zastínili staré bojové pole, na kterém však stále probíhá bitva o přístup a ochranu paměti. V tomto článku se pokusíme shrnout a prezentovat techniky útoků a obran na aplikace a operační systémy typu cokoliv-overflow. Článek vychází převážně z publikací konference BlackHat 200/72008/2009.

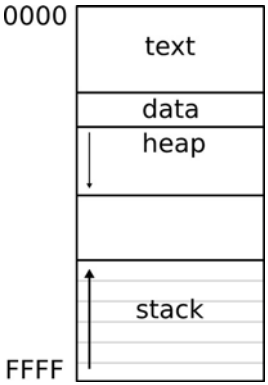
Klíčová slova: buffer overflow, call stack, fandango on core, AlephOne, memory protection, /GS, SafeSEH, DEP, PaX, W^X, ASLR, GrSecurity

Hned v úvodu bych rád upozornil čtenáře, že všechen následující materiál byl více či méně převzat z uvedené literatury a neobsahuje žádný vlastní výzkum. Článek by měl posloužit zájemcům jako rozcestník pro danou problematiku a případně aktualizaci znalostí.

O tématu stack buffer overflow bylo napsáno již velmi mnoho materiálu, proto zde uvedeme pouze krátkou rekapitulaci. Následující část osvětlí základní stavební prvky této problematiky a důsledky zneužití chyb tohoto typu. Předpokládejme, že cílem našeho snažení je získání kontroly nad procesem pomocí zneužití chyby v programu. Případné zájemce o hlubší průzkum úvodní problematiky viz [1].

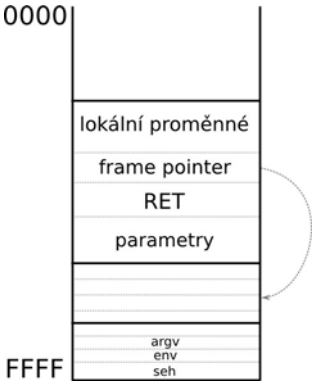
Vyrovnávací paměť, přetečení a zásobník

Buffer je spojitý blok paměti, který obsahuje pole prvků jednoho datového typu, ne vždy nutně pole znaků. O **přetečení/overflow** bufferu hovoříme v případě, že při práci s výše zmíněným bufferem dojde k zápisu do paměti za jeho hranice. Rozložení částí programu v adresním prostoru (u počítačů s von Neumanovou architekturou) procesu ukazuje obrázek 1. Text představuje kód programu, data jeho statická data, heap jeho dynamicky alokovaná data a stack jeho zásobník.



Obr. 1 Rozložení adresního prostoru procesu

V dnešních počítačových systémech jsou programy strukturovány do procedur a funkcí které se navzájem volají. Pro udržení informací o průběhu programu, tj. zanořování jednotlivých funkcí, používají programy k uložení potřebných dat a metadat paměťový prostor nazývaný **stack**. Stack je používaný pro: předávání parametrů, ukládání návratových adres (RET/%EIP), uložení pointerů potřebných pro zřetězování/řízení stacku a práci s jeho daty (SFP/%EBP), uložení lokálních proměnných funkcí, na Windows navíc pro ukládání adres obsluh výjimek.



Obr. 2 Data na zásobníku

Volání funkce programu má zpravidla 3 části:

- *prologu* – části, kde probíhá příprava parametrů volání funkce, uložení adresy na které má program pokračovat po návratu z funkce, zapamatování

ukazatele na aktuální stack frame a vytvoření prostoru pro lokální proměnné volané funkce,

- *těla funkce* – samotného těla funkce,
- *epilogu* – části, kde se řízení programu vrací do volající funkce pomocí uložené adresy RET a obnovení stacku volající funkce.

Protože v současných počítačích není oddělen adresní prostor řídících dat na stacku (SFP, RET) a lokálních dat programu/funkce, může za určitých okolností dojít k přepsání řídících dat (adresy RET) ...

```
---------------------------------------
void function(char *str) {
        char buffer[32];

        strcpy(buffer,str);
}

void main(int argc, char *argv[]) {
        function(argv[1]);
}
---------------------------------------
```

Obr. 3 Ukázka programu s chybou buffer overflow

Ve výše uvedeném příkladu je nejjednodušší případ pro stack overflow. Pokud bude jako parametr programu předán řetězec větší než 32 znaků, dojde na stacku k přepsání návratové adresy (i frame pointeru) a program, při návratu z funkce, skončí skokem do neznáma... Útočník se může pokusit vložit takový řetězec, který do RET vloží nějakou smysluplnou hodnotu, aby tak program pokračoval na zvoleném místě a udělal něco pro útočníka ... například spustil *execve(/bin/sh)*, tj. nahradil současný proces shellem, aby mohl útočník spouštět další příkazy. Pokud by byl výše uvedený program označen jako *setuid*, získal by výsledný shell příslušná práva.

Vkládanému řetězci se zpravidla říká **shellkód** a musí splňovat několik parametrů v závislosti na použití:

- musí být zapsán správně pomocí instrukcí pro daný procesor,
- neměl by obsahovat žádné absolutní adresy (PIC),
- neměl by obsahovat znaky `0x0`, které neprojdou přes funkce, které zpracovávají řetězce.

```
---------------------------------------
int main(void) {
        char buf[] = "Hello world!\n";
        write(1, buf, sizeof(buf));
        exit(0);
}
---------------------------------------
```

Obr. 4 Předlohla pro shellkód

```
hAAAAX5AAAAHPPPPPPPPPah0B20X5Tc80Ph0504X5GZBXPh445AX5XXZaPhAD00X5wx
xUPTYII19hA000X5s0kkPTYII19h0000X5cDi3PTY19I19I19I19h0000X50000Ph0
A0AX50yuRPTY19I19I19I19h0000X5w100PTYIII19h0A00X53s0kPTYI19h0000X5
0cDiPTYI19I19hA000X5R100PTYIII19h00A0X500yuPTYI19I19h0000X50w40PTY
II19I19h0600X5u800PTYIII19h0046X53By9PTY19I19I19h0000X50VfuPTYI19I
19h0000X5LC00PTYIII19h0060X5u79xPTY19I19I19I19h0000X5000FPTY19I19h
2005X59DLZPTYI19h0000X500FuPTYI19I19h0010X5DLZ0PTYII19h0006X50Fu9P
TY19I19I19I19h0000X5LW00PTYIII19h0D20X5Lx9DPTY19h0000X5000kPhA0A0X
5ecV0PTYI19I19h0B0AX5FXLRPTY19h5550X5ZZZePTYI19ÿä
```

Obr. 5 Výsledný ascii shellkód s aplha loaderem [17]

V reálných případech není situace tak jednoduchá jako v našem příkladu. Pro úspěšné zneužití potřebuje útočník: jakkoli nahrát vlastní kód do paměti programu, případně najít adresu kde se užitečný kód již nachází a přepsat zásobník tak, aby RET začal ukazovat do místa kam byl shellkód vložen.

Nicméně zjistit správnou adresu nemusí být vždy jednoduché ani pro jednoduché programy, nehledě na to, že různé kompilátory používají různá zarovnávání a optimalizace, takže na různých počítačích může být stack frame zneužívané funkce ve velmi odlišných místech. Tento fakt se dá obejít několika způsoby:

- zvětšením shellkódu pomocí několika instrukcí NOP, tím zvětšíme rozsah paměti, kam může RET ukazovat,
- pokud je buffer příliš malý a můžeme kontrolovat proměnné prostředí, můžeme shellkód vložit do nich, protože jsou vždy na začátku stacku a mohou být *libovolně* veliké,
- pokud ani na to není prostor je možné použít návrat na nějaký registrový skok (jmp %reg),

- můžeme skákat na adresu funkce z některé sdílené knihovny (ret2lib),
- případně do textového segmentu samotného programu.

Některé zajímavé příklady z praxe ukazují, že tento typ chyb je i přestože velmi starý, stále rozšířený a mezi útočníky oblíbený:

- Morris worm – pro šíření zneužíval chybu v unixovém démonu finger (1988)
- Slammer worm – se šířil pomocí chyby v Microsoft's SQL serveru (2003)
- Blaster worm – šířil se díky chybě v Microsoft DCOM (2003)
- Witty worm – pro šíření zneužíval chybu ve firewallu Internet Security Systems (2004)
- Conficker A – pro šíření využíval chybu v RPC (MS08-067; 2008)

Obrany a jejich obcházení

Proti výše uvedené chybě bylo vynalezeno několik více či méně účinných ochran, všechny však byly poraženy velmi záhy po svém uvedení.

Odstranit příčinu

První možností která se samozřejmě nabízí je naučit programátory nedělat chyby – což je nemožné. Druhou je pokusit se používat zabezpečené varianty problematických funkcí (*libsafe*), buffer overflow však nenastává pouze při práci s řetězci. Dále se nabízí statická kontrola kódu, ale ani *splint* ani *valgrind* není všemocný, nehledě na to že by jejich použití muselo být běžnou součástí vývojového procesu.

Existují i rozšíření překladačů (gcc FORTIFY_SOURCE), které kontrolují práci s buffer při překladu, tato ochrana funguje pouze pro některé programové konstrukce. Též je možné kontrolovat programy za běhu (bounds checking), tato technika je však výkonnostně náročná.

Také bychom mohli používat pouze jazyky, které nepodporují přímé ukazatele do paměti (Java, Python, Ruby), ale kdo by chtěl psát operační systém v Jave.

NX/XD/XN/DEP – nespustitelné stránky

Myšlenka jednoduchá, rozšířit metadata stránek o další bit (RW > RWX) a označit paměťové stránky stacku jako nespustitelné. Pokud do takové stránky umístí útočník svůj kód a pokusí se jej vykonat procesor mu to nedovolí. Tato vlastnost je do procesorů implementována od zhruba od roku 2004, ale i před tím byly projekty (PaX, ExecShield, W^X) které to dokázali softwarově emulovat...

Implementace této ochrany vyžaduje jak podporu v operačním systému, tak i ve spouštěném programu. Ne všechny jsou ale schopné s tímto korektně fungovat. Např: Java, Lisp, Xserver, ... používají stack pro dynamické vytváření a spouštění kódu (trampolíny, ...) a proto pro ně musí být vytvořeny výjimky.

Softwarové verze této ochrany lze někdy vypnout odskokem (pomocí RET) přes volání *mprotect()*, které s nastavením zabezpečení stránek manipuluje.

Windows XP mají podporu NX od SP2 s defaultní politikou *OptIn* (stejně tak Vista), která říká že DEP je zapínáno na požádání, Windows 2003, 2008 mají politiku *OptOut*. Na 64bitových Windows jsou již všechny programy se zapnutým DEP, který nejde vypnout, ale Internet Explorer zde zůstává stále 32b proces a podléhá výše uvedeným politikám a lze tuto ochranu vypnout [2]. Stejně tak DEP nepodporoval Firefox 2.

Hlavním způsobem jak obejít tuto ochranu je však technika return-into-libc [3]. Pokud není možné vykonávat kód uložený na stacku, není nic jednoduššího než použít již existující kód, který je určen ke spuštění – kód sdílených knihoven nebo samotného programu. Oblíbenou technikou je tedy příprava parametru na stack a skok do funkce *system()* z knihovny libc. Příprava stacku pro tuto funkci může vyžadovat zvětšení parametru[1].

V případě nových procesorů x86_64 se předávání parametrů volání provádí nikoliv pamětí na stacku ale registry procesoru, toho je možné dosáhnou technikou *ret code chunking – gadgets* [4, 8, 5]. Nejdříve odskočit na nějaké místo v paměti, kde se nacházejí instrukce typu *'pop %rdi; ret'* které natáhnou hodnotu ze stacku do registru a až poté odskočit do funkce *system()*. Takto je možné zřetězit více potřebných instrukcí nebo funkcí.

ASLR – prostor plný náhody

Jak je vidět, jednu z hlavních rolí v buffer overflow hraje znalost adres kódu nebo dat, které útočník potřebuje, ať už se jedná o jím vložený shellkód, nebo adresy užitečných funkcí. Před ASLR byly všechny části programu, sdílené knihovny i stack mapovány na stejná místa ve virtuálním adresním prostoru procesu. Řešením by tedy mohlo být, změnit toto statické mapování tak, aby měl každý proces knihovny, kód, data i stack pokaždé na jiném místě v paměti. To by mělo

[1] Např: /////////////./////////bin/sh

znemožnit útočníkovi nalézt potřebné adresy. Jako první navrhl a implementoval tuto myšlenku projekt PaX v roce 2001 [9].

V Linuxu je od jádra verze 2.6.12 znáhodněna část virtuální adresy, 8 bitů pro x86_32 a 28 bitů na x86_64 [5]. Windows podporují tuto ochranu od verze Vista, např. haldu znáhodňují v rozsahu 2 MB. Implementace ASLR na Mac OS X je v plenkách, znáhodňují se adresy knihoven, ale už ne dynamického linkeru ze kterého lze pak informace získat [10].

Bohužel, kód programu, musí být přizpůsoben k relokaci a mnoho softwaru dodnes není (např. pluginy browserů). Útočník může vložit do stránky takový EMBED objekt, který způsobí natažení nerelokovatelného pluginu a potom jej zneužít za znalosti adresy kam se nahrál. Stejně tak implementace nemusí být bez chyby, v linuxovém jádře ($< 2.6.20$) nebyla znáhodněna adresa na kterou se mapovala knihovna *linux-gate.so* – knihovna, která poskytuje programu možnost volat jádro bez použití přerušení (které je pomalé). Tato chyba umožnila provádět return-into-libc i na systémech s ASLR. Navíc, není v linuxu defaultně znáhodněno mapování samotného programu.

Linuxových 8 bitů (pro x86_32) není dnes příliš a tak se dá adresa odhadovat i hrubou silou [7]. Volání *fork()* totiž zachovává náhodnost adresního prostoru a tak je zde prostor pro pokusy správnou adresu odhadnout. Proti tomuto útoku zavedl projekt GrSecurity+PaX znemožnění volání *fork()* pokud proces spadl vícekrát než je za časový úsek zdrávo, navíc používá větší míru znáhodnění adresy (podobně jako ExecShield).

Další možností [2] je sprejování. Pokud může útočník alokovat dostatečné množství místa na shellkód (řádově megabajty až stovky megabajtů), může zvýšit své šance pro úspěšné odhadnutí potřebné adresy. Manipulace s proměnnými prostředí nebo používání jazyků jako JavaScript, Java, Flash tyto alokace dovolují. Na obrázku 6 je ukázka z exploitu pro Internet Explorer [18].

Metodou částečných přepisů (partial overwrites), může útočník zachovat základ adresy, tu část ve které je znáhodnění, a přepisem 1 nebo 2 nejméně významných bitů přesto dosáhnout potřebných manipulací s pamětí na základě znalosti relativního umístění objektů v paměti.

Pokud všechny výše uvedené metody selhávají může se útočník pokusit znáhodnění prostoru odhalit pomocí chyb formátování řetězců.

Zřejmě nejnovější technika [6] zneužívá další chyby implementace ASLR v současném linuxovém jádře. Zde je náhodnost rozložení závislá na PID procesu a *čase*. Při svém výzkumu Fritsch zjistil, že existuje časové okno veliké až 2 minuty, kdy dokáže útočník nasimulovat stejná, nebo velmi podobná, znáhodnění pro různé procesy za pomoci manipulace s PID, které dokáže ovládat pomoci volání *fork()*, *execve()* a *usleep()*. S pomocí získaných hodnot pak zjistit adresy potřebné pro úspěšné zneužití buffer overflow, to za použití několika málo pokusů (na platformě x86_64).

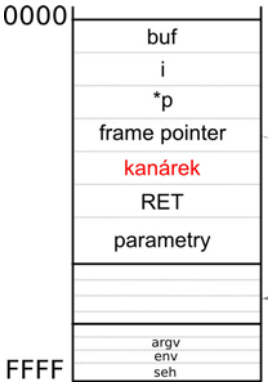
```
--------------------------------
...
var shellcode = unescape("%ue8fc%u0044%u0000%u458b%u8b3c%u057c
                         %u0178%u8bef%u184f...");
var block = unescape("%u0c0c%u0c0c");
var nops = unescape("%u9090%u9090%u9090");


while (block.length < 81920) block += block;
var memory = new Array();
var i=0;
for (;i<1000;i++) memory[i] += (block + nops + shellcode);

document.write("<iframe src=\"iframe.html\">");
</script>
</html>

<!-- iframe.html
<XML ID=I><X><C><![CDATA[
        <image
                SRC=http://&#3084;&#3084;.xxxxx.org
        >
]]></C></X></XML>
---------------------------------------
```

Obr. 6 Ukázka exploitu MS IE XML Parser Remote Buffer Overflow

A nakonec Kanárek

V roce 1997 přišel Cristin Cowan s dalším nápadem (*StackGuard*), v prologu funkce umístit na stack mezi lokální proměnné a návratovou adresu nějakou hodnotu/kanárka/cookie a v epilogu, před skokem na RET, zkontrolovat, zda nebyla změněna. Pokud by došlo v průběhu funkce k útoku, byla by tato hodnota přepsána a útok odhalen [13].

Zhruba 24 hodin po zveřejnění projektu se ukázalo, že statické hodnoty nejsou žádnou reálnou ochranou a buď se dají odhadnout hrubou silou, nebo z paměti přečíst či přepsat jejich předlohu. Postupem času byla technika této ochrany vylepšena na náhodný XORovaný kanárek, který se velmi špatně odhaduje, případně kanárek který se nedá kopírovat funkcemi pro práci s řetězci (tzv. terminator cannary – NULL(0x00), CR (0x0d), LF (0x0a) a EOF (0xff)).



Obr. 7 Ukázka zásobníku s kanárkem

Nicméně, původní implementace nechránila frame pointer, pomocí jehož přepisu (nebo přepisu jeho části) bylo dále možné manipulovat se stackem a lokálními proměnnými volající funkce (caller's frame pointer) [14]. Později byl kanárek posunut (první Microsoft/GS, posléze i gcc) mezi frame pointer a lokální proměnné.

I přesto však zbývá prostor pro útok, útočník může stále přepisovat lokální proměnné, což v případě přepsání pointeru může vést k zápisu na libovolnou adresu v paměti podobně jako u heap overflow. Tímto způsobem se může útočník pokusit přepsat pointer na funkci, která je později za jeho pomoci volána nebo se může pokusit přepsat GOT/PLT [16], tabulku která drží informace o adresách relokovaných knihovních objektů a funkcí, které jsou použity ještě před či dokonce při kontrole kanárka a nebo obecně, dosáhnout zápisu na vybrané místo v paměti.

Na podobném principu pracuje zneužití handlerů výjimek. Na systémech Windows, jsou na stacku uloženy i adresy obsluhy výjimek (SEH), ty jsou sice uloženy až za kanárkem a měly by tedy být chráněny, ale pokud nastane ve funkci výjimka (např. při přetečení nějakého counteru s následným přístupem do nealokované paměti), dojde k použití této adresy ještě před kontrolou kanárka. Windows se snaží implementovat SafeSEH, tj. mít v paměti seznam bezpečných obsluh výjimek, ale opět je tato vlastnost zapnuta pouze v případě plné podpory DEP a SEH v programu a mnoho knihoven tuto ochranu dosud nepodporuje.

Kolem roku 2004/2005 byly kompilátory obohaceny o schopnost přeskládávat lokální proměnné tak, aby byly zneužívané buffery uloženy před ostatní lokální proměnné a navíc doplnit stack o kopii parametrů funkce (projekt *ProPolice*). Pokud tedy dojde k přetečení některého z bufferů, neovlivní to lokální proměnné, ani parametry funkce a navíc dojde k přepsání kanárka, což je při návratu z funkce detekováno a proces je ukončen.

```
-----------------------------------
/* sg1.c *
* specially crafted to feed your brain by gera@corest.com */
int func(char *msg) {
        char buf[80];
        strcpy(buf,msg);
        // toupper(buf); // just to give func() "some" sense
        strcpy(msg,buf);
}

int main(int argv, char** argc) {
        func(argc[1]);
}
-----------------------------------
```

Obr. 8 Zápis na vybrané místo v paměti [14]

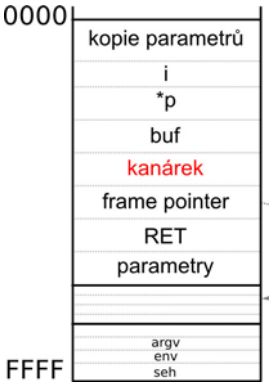
Práce a kanárkem, ale přidává do programů nezanedbatelnou režii, proto nebývalo jejich používání implicitně zapnuto a navíc jsou tímto způsobem chráněny pouze některé funkce, zpravidla ty které používají znakové buffery o délce větší než 4. Stále tedy zbývá dostatek funkcí, které chráněné nejsou a přesto se dají zneužít (CVE-2007-0038, CVE-2006-3747).

V neposlední řadě může, podobně jako u ASLR, dojí chyb formátovacích řetězců či nechráněných frame pointerů k vyzrazení hodnoty kanárka a tudíž eliminaci této ochrany. Snad poslední chybou do výčtu chybí, byla implementační část v GCC, kde je inicializační část kanárka závislá na statických hodnotách a čtení náhodných dat z */dev/random* bývalo z výkonnostních důvodů defaultně vypnuto [5]. Distribuce Debian zapnula tuto ochranu ve verzi 5.0/Lenny.

Alternativou ke StackGuardu je StackShield, ten klonuje návratovou adresu do místa v paměti které se nedá přepsat, a před návratem z funkce jí vrací na původní místo bez porovnání. Tato ochrana je celkem silná, nicméně stále neposkytuje ochranu proti manipulaci s lokálními proměnnými, argumenty nadřazené funkce a přepis GOT, navíc nedetekuje pokusy o útok, pouze částečně eliminuje jejich dopad [16].

Závěr

V tomto článku jsem se pokusil shrnout metody, které byly navrženy a implementovány jako obrana proti chybám typu stack buffer overflow. Na většinu z nich byl nalezen protiútok velmi záhy po uvedení ochran. Podle výzkumu který provedl Hagen Fritsch [5], jsou ochrany zásobníků pomocí kanárků zatím nejúčinnější, ale hlavně v kombinaci s ostatními technikami ASRL a NX.



Obr. 9 Ukázka stacku při použití ProPolice

Mohlo by se zdát, že hlavní chybou je použití von Neumanovy architektury, ale nejnovější výzkumy ukazují, že se dají nalézt techniky pro permanentní vkládání a spouštění kódu i na platformách založených na Harvardské architektuře [15].

Reference

[1] *Aleph One*: Smashing The Stack For Fun And Profit, Phrack 49.

[2] *Alexander Sotirov, Mark Dowd*: Bypassing Browser Memory, Black Hat 2008.

[3] *Solar Designer*: Getting around non-executable stack (and fix).

[4] *Nergal*: The advanced return-into-lib(c) exploits: PaX case study, Phrack 58.

[5] *Hagen Fritsch*: Stack Smashing as of Today: A State-of-the-Art Overview on Buffer Overflow Protections on linux_x86_64, Black Hat, 2009.

[6] *Hagen Fritsch*: Bypassing ASLR by predicting a process randomization, Black Hat, 2009.

[7] *Hovav Shacham et al.*: On the Effectiveness of AddressSpace Randomization http://www.cs.jhu.edu/∼rubin/courses/fall04/asrandom.pdf

[8] *Hovav Shacham*: *The Geometry of Innocent Flesh on the Bone: Return-into-libcwithout function calls (on the x86)* http://www.cs.jhu.edu/∼rubin/courses/fall04/asrandom.pdf

[9] Homepage of The PaX Team http://pax.grsecurity.net/

[10] *Charlie Miller, Vincenzo Iozzo*: Fun and Games with Mac OS X and iPhone Payloads, Black Hat, 2009.

[11] *M. Dobšíček, R. Ballner*: Linux bexpečnost a exploity, Kopp, 2004.

[12] *Pavel Herout*: Učebnice jazyka C, Kopp, 1994.

[13] *Shawn Moyer*: (un)Smashing the stack, Black Hat, 2007.

[14] *Gerardo Richarte*: Four different tricks to bypass *StackShield* and *StackGuard* protection
http://www.coresecurity.com/files/attachments/StackguardPaper.pdf

[15] *Aurelien Francillon, Claude Castellucia*: Code injection attacks on Harvard-Architecture devices
http://planete.inrialpes.fr/∼ccastel/PAPERS/CCS08.pdf

[16] *wilder* exploitovanie cez plt (procedure linkage table) a got (global offset table)
http://www.hysteria.sk/prielom/15/#7

[17] ShellForge http://www.secdev.org/projects/shellforge

[18] MS Internet Explorer XML Parsing Remote Buffer Overflow Exploit
http://www.milw0rm.com/exploits/7410

Link load balancing i pro BFU

Jan Ježek

E-mail: jjezek@kerio.com

Abstrakt

Článek popisuje funkci rozdělování zátěže mezi více internetových připojení tak, jak je implementována ve firewallu firmy Kerio Technologies. Rozebere přístup k řešení takto relativně složitého technického problému tak, aby mu porozuměl i Běžný Franta Uživatel. Provede vás cestou jak vznikala zmiňovaná funkce se zaměřením na zajímavé technické problémy, na které se během vývoje naráželo.

Co je link load balancing

Pojmem link load balancing v Kerio WinRoute Firewallu chápeme schopnost využít kapacity několika internetových připojení a dynamické rozdělování zátěže mezi tyto připojení. Ta mohou být zcela heterogenní od různých poskytovatelů a na různých technologiích, ať už je to pronajatá linka, kabelové připojení, ADSL, ISDN, apod.

Předchozí odstavec v podstatě přesně vystihuje to, co jsme věděli zcela na začátku vývoje této funkce, nic víc, nic míň. Dále se budu věnovat cestě od této hrubé definice k reálné a fungující implementaci. Pro upřesnění ještě dodám, že nás zajímaly pouze protokoly z rodiny IPv4, nikoliv IPv6, kde by byla situace jiná a zřejmě o poznání jednodušší. Cesta k cíli byla ve skutečnosti o něco složitější a méně vodopádovitá, než je zde popsáno, ale podstata je, myslím, vystihnuta.

Studie proveditelnosti

Na to, abychom mohli rozkládat zátěž mezi více linek, potřebujeme směrovat provoz do každé linky podle jejího aktuálního vytížení. Z pozice, ve které se firewall nachází, však dokáže směrovat pouze odchozí provoz, tj. z vnitřní sítě ven do internetu, příchozí provoz, a o ten nám jde především, směrovat nemůže. Se

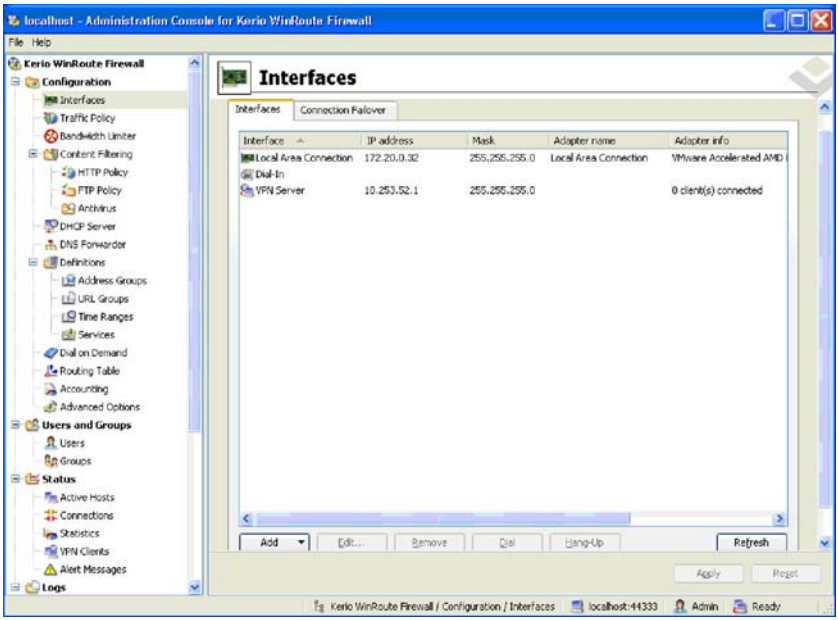
spoluprácí ze strany poskytovatelů nelze počítat, směrovací protokoly nebývají na koncových přípojkách podporovány.

Vycházejme z toho, že firewall vidí veškerá odchozí spojení do internetu a že v 99,9 % (kvalifikovaný odhad, nikoliv však přesné číslo) případů nasazení je použita technologie překladu síťových adres, NAT. Budeme-li tedy spojení překládat na IP adresu některého z poskytovatelů internetu, dosáhneme toho, že i příchozí provoz bude směrován přes příslušného poskytovatele a tedy i přes příslušnou linku. Rozložení zátěže pak dosáhneme střídáním poskytovatelů, ať už jednoduše stylem round-robin a nebo sofistikovaněji, např. podle celkové kapacity a aktuálního vytížení.

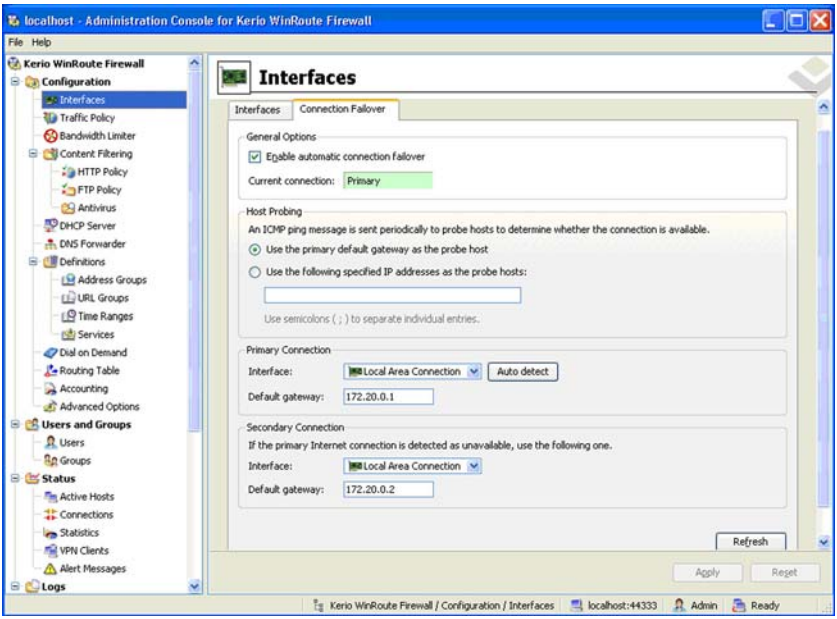
Touto jednoduchou metodou je tedy technicky možné funkci link load balancing implementovat bez jakýchkoliv nestandardních nároků na typ připojení.

Jak budeme funkci prezentovat uživateli

Kerio WinRoute Firewall „odjakživa“ umí pracovat s jedním internetovým připojením, ať už pevným nebo s vytáčeným na žádost. Ano, vytáčení na žádost se i dnes ve speciálních případech stále používá. Dále existuje možnost definovat druhé připojení, ale pouze jako záložní pro případ výpadku primární linky.



Obr. 1



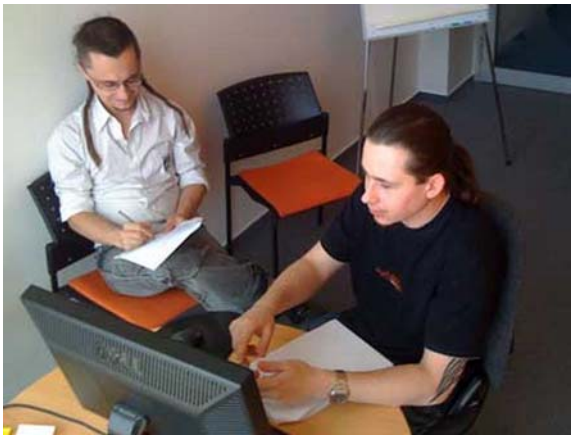
Obr. 2

V uživatelském rozhraní jsou dvě relevantní obrazovky. Jedna zobrazuje seznam síťových rozhraní s příslušnými stavovými informacemi jako je IP adresa, brána, atp. (obr. 1) a na druhé z nich se zapíná a nastavují se parametry záložní linky.

Naše nová funkce logicky bude zapadat do těchto obrazovek, otázka ovšem je jak, aby to uživatel co nejlépe pochopil a hlavně aby to bez školení a studia návodu dokázal nastavit a používat. Není reálné pouze přidat další nastavení, protože už v tuto chvíli je ovládacích prvků přespříliš a jsou komplikované.

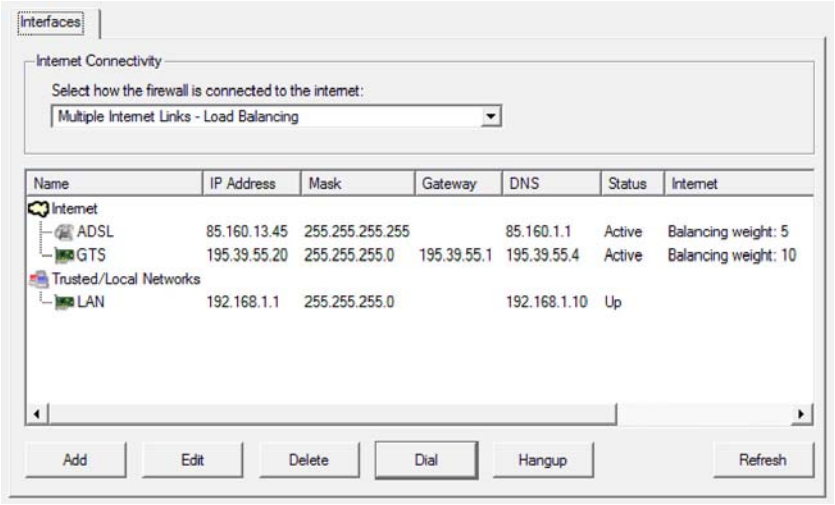
Následuje fáze, ve které jsme dělali design uživatelského rozhraní. Já zde vyprávění poněkud zestručním a rovnou řeknu, že návrhů postupně vzniklo několik a vítězným se stal až ten s pořadovým číslem 11.

Jednotlivé návrhy byly podrobovány tzv. usability testům. Ty probíhají ve stručnosti tak, že jeden člověk v roli laboratorní myši se posadí před prototyp aplikace a je mu moderátorem zadán určitý úkol, který musí splnit (obr. 3). V našem případě to bylo nastavení link load balancingu. Moderátor ani nikdo jiný během testu nijak aktivně nezasahuje, pouze sleduje jak daný člověk reaguje na rozhraní aplikace a jak se mu daří úkol splnit. Výsledky se pak vyhodnotí, rozhraní se poupraví a nebo v případě velkého neúspěchu kompletně přepracuje. To celé se několikrát opakuje.



Obr. 3

Náš výsledek je vidět na obr. 4:



Obr. 4

Ze dvou obrazovek vznikla jediná. V horní části je volba typu konektivity (jedna linka, záložní linka, link load balancing) a v tabulce, kde původně byl jen pasivní výpis síťových rozhraní, lze nyní přímo nastavovat parametry jak záložní linky tak i rozkládání zátěže. Od skutečnosti v později vydaném produktu se tento návrh již prakticky neliší.

Můžeme programovat...

Tady bych mohl skončit, protože programování bývá obvykle už to nejjednodušší na celé práci na nové funkci. Nicméně tak, jak to obvykle bývá, nic není tak růžová, jak se na první pohled může zdát. I v tomto případě totiž nastal efekt, kterému v Keriu říkáme „betonová zeď“. Z ničeho nic přijde poznání, že tak, jak si to člověk naplánoval, to není možné realizovat.

Co se stalo? Jako každá firma vyvíjející tzv. krabicový software provozujeme prakticky jediný skutečně efektivní model testování, a sice testování sami na sobě. V angličtině pro to je pěkný výraz „eat your own dogfoot“. Nedlouho potom, co jsme první alfa verzi link load balancingu nasadili a pořídili jsme si na to k naší gigabitové lince dvě ADSL připojení, začali se trousit kolegové s problémy:

- „Nějak mi v poslední době blbne web.“
- „Neděláte s tím něco? Nefunguje mi video na YouTube.“
- „Vůbec mi nefunguje přístup na e-mail přes web.“
- „Odhlašuje se mi webový ICQ klient.“

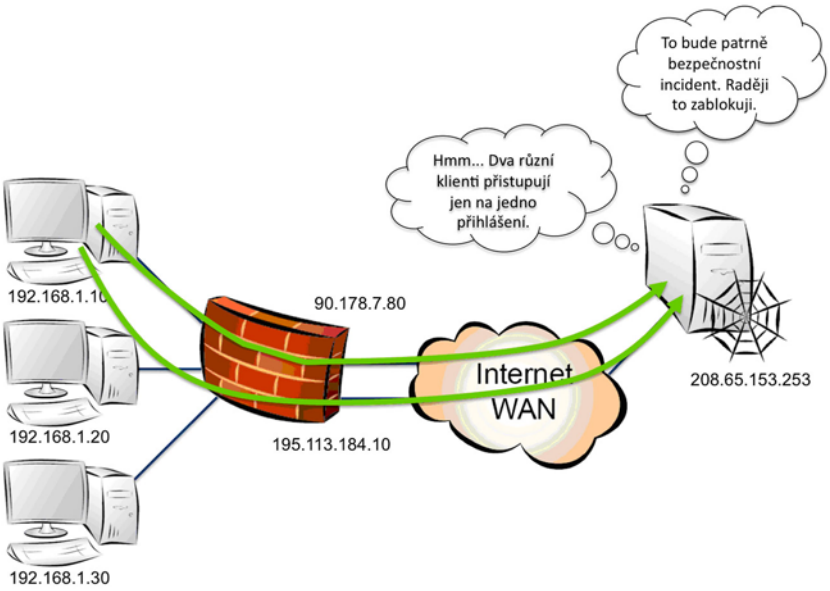
Po tom, co jsme eliminovali možnost nějaké triviální chyby v kódu, jsme museli připustit, že chyba bude už patrně v samotném návrhu. Webové aplikace (principielně se problém ale nemusí omezovat jen na ně) totiž typicky v rámci jednoho sezení vytvoří desítky i stovky TCP spojení. Pokud pro každé z nich je naším algoritmem rozhodnuto, že se pošle přes jinou linku a tudíž z jiné IP adresy, může dojít k tomu, že server takové spojení odmítne, protože z jeho pohledu pochází od dvou různých klientů. Může se domnívat, že došlo k odcizení session cookie, a spojení ukončí (viz obr. 5).

Pokus o řešení č. 1

Prvním nápadem jak danou situaci řešit bylo svázat rychle po sobě jdoucí spojení mezi týmž klientem a týmž serverem s jednou internetovou linkou. Toto řešení pomohlo pro většinu webových aplikací. Nicméně např. YouTube a zmiňovaný webový ICQ klient stále nefungovali správně.

Pokus o řešení č. 2

Zkoumáním jsme zjistili, že ICQ klientovi nestačí to, co my považujeme za „rychle po sobě jdoucí“. Zkusili jsme tedy prodloužit interval, v němž zůstává pro stejného klienta a stejný server linka konstantní, až na minuty, konkrétně 15 minut.



Obr. 5

Samozřejmě, že toto řešení již citelně ovlivňuje efektivitu algoritmu pro rozdělování zátěže, zejména pro menší sítě. Bohužel, zákazník neocení vysoce efektivní algoritmus, který mu však znefunkčňuje jeho aplikace.

ICQ se nyní přestalo odpojovat, ale co stále nefungovalo, byla videa z YouTube.

Řešení č. 3

Důvod se ukázal být prostý. YouTube je totiž z celkem pochopitelně distribuovaná aplikace. Z jiného serveru stahuje HTML kód svých stránek, z jiného kód flashového přehrávače videa a z jiného vlastní video data. Pokud však kdykoliv v tomto procesu dojde ke změně IP adresy klienta, dojde k chybě a video se nezobrazí.

Pro nás to znamená to, že musíme svazovat s konkrétní linkou nikoliv dvojici klient-server, ale dokonce pouze samotného klienta. Jinými slovy, každý klient v lokální síti bude v rámci 15 minutových intervalů vždy svázán s jednou internetovou linkou a nebude přepínán na jinou. Efektivita algoritmu se tím ještě dále snižuje. Specielně v malých sítích pouze o několika počítačích může dojít k nevhodnému rozložení klientů mezi jednotlivá internetová připojení.

Happy End

Protože ponechat výslednou funkci tak neefektivní se nám nezdálo vhodné, rozhodli jsme se, že v aplikaci umožníme uživatelsky zvolit režim, ve kterém se balancování bude provádět. Každý režim má svá pozitiva a negativa. V tabulce 1 je vidět, jaké režimy to jsou:

Tabulka 1

Režim	**Efektivita**	**Kompatibilita**
ruční	–	vysoká
automatický dle počítačů	nízká	vysoká
automatický dle spojení	vysoká	nízká

Ve výchozím stavu, nenastaví-li uživatel v konfiguraci nic, se použije balancování automatické dle počítačů, tj. algoritmus sice s relativně nízkou efektivitou, zejména pro malé sítě, ale s vysokou kompatibilitou s online aplikacemi (obr. 6).

Name	Source	Destination	Service	Action	Translation
NAT Setting	Trusted/Local	Internet	Any	✓	NAT

Obr. 6

V případě, že si to uživatel přeje, může pro konkrétní provoz, např. pro VoIP aplikace použít ruční režim, ve kterém konkrétní protokolům, např. SIP, přiřadí natvrdo konkrétní internetovou linku bez žádného automatického balancování. Je to taková obdoba policy routingu (obr. 7).

Name	Source	Destination	Service	Action	Translation
Fred uses ADSL1	fred	Internet	Any	✓	NAT (ADSL1)
Browsing via ADSL2	Trusted/Local	Internet	HTTP HTTPS	✓	NAT (ADSL2)
Other traffic via ADSL1	Trusted/Local	Internet	Any	✓	NAT (ADSL1)

Obr. 7

Naopak pro provoz, o kterém jsme si jisti, že nemá problémy s kompatibilitou při měnící se IP adrese klienta, může uživatel nastavit, aby se balancování provádělo pro každé spojení zvlášť (obr. 8).

Name	Source	Destination	Service	Action	Translation
NAT Setting	Trusted/Local	Internet	Any	✓	NAT Balancing Per Connection

Obr. 8

Na závěr si dovolím uvést jeden strip z Dilberta, který, myslím, pěkně ukazuje pozici vývojáře, řeší-li rozpor složitost vs. snadnost použití (obr. 9):

Obr. 9

Vulnerabilities – zranitelnosti

Aleš Padrta

E-mail: apadrta@civ.zcu.cz

Abstrakt

Se zranitelnostmi (anglický termín vulnerabilities) se členové CSIRT, ale také administrátoři i běžní uživatelé, setkávají téměř na každém kroku. Jedná se o slabá místa systémů, kterým musí být věnována dostatečná pozornost, jinak je mohou potenciální útočníci použít pro napadení daného systému. Příspěvek se komplexně shrnuje problematiku od definice pojmu zranitelnosti a jeho kontextu, přes klasifikaci a životní cyklus, až po informační zdroje, které se zabývají identifikací a popisem zranitelností. Součástí jsou také příklady konkrétních zranitelností.

Abstract

CSIRT members, system administrators and also user have encountered many vulnerabilities in their lives. A vulnerability means some week parts of systems, which can be used to attack that system. The paper is concentrating on vulnerability description, its context, classification, life cycle, and information source related to vulnerability identification and description.

1 Úvod

Setkávání se se zranitelnostmi je nedílnou součástí profesního života všech členů CSIRT a administrátorů systémů, ale také běžných uživatelů. Vulnerabilites, jak jsou zranitelnosti nazývány v anglickém jazyce, představují slabá místa systémů, kterým musí být věnována dostatečná pozornost. V opačném případě je mohou potenciální útočníci použít pro napadení daného systému. Tento článek si klade za cíl komplexně seznámit čtenáře s problematikou zranitelností od obecného úvodu až ke konkrétním příkladům.

V kapitole 2 je věnována pozornost vlastní definici zranitelnosti a také jejímu kontextu. Kapitola 3 seznamuje s různými typy zranitelností a jejich klasifikací. V kapitole 4 je následně popsán životní cyklus zranitelnosti od jejího vzniku až po její eliminaci. Kapitola 5 představuje informační zdroje týkající se zranitelností a stručný návod jak s těmito informacemi nakládat. Vztah CSIRT ke zranitelnostem je představen v kapitole 6. Na závěr jsou v kapitole 7 popsány vybrané konkrétní zranitelnosti.

2 Definice zranitelnosti

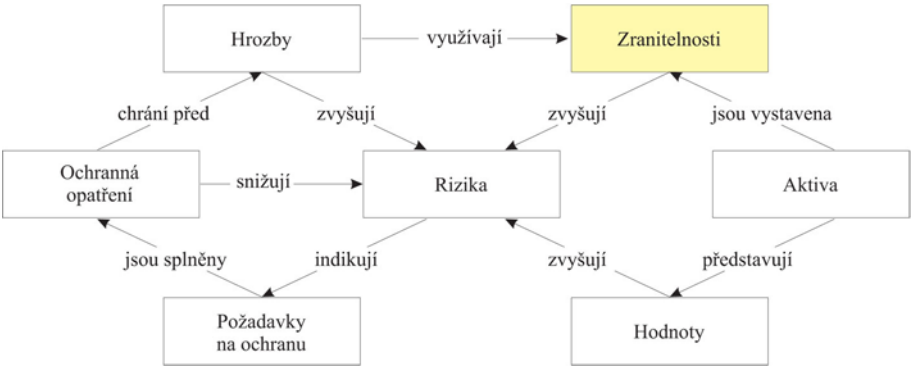
Pojem zranitelnost (anglicky vulnerability) je v prostředí IT relativně známý pojem. Slouží k označení slabých míst, která mohou být zneužita k porušení integrity a narušení bezpečnosti systémů. Zpravidla se jedná o možnost zneužít nějakou funcionalitu nebo chybu systému. V některých kruzích je za zranitelnost také považována neznalost uživatele.

Pro úvodní představu o *roli zranitelnosti* při útoku slouží obrázek 1. V levé části jsou naznačeny aktivity, které útočník provádí k dosažení svého cíle, napravo pak konkrétní příklad. Vhodně vybraná zranitelnost je nedílnou součástí útoku [1]. Valná většina zranitelností je však zneužitelná pouze za velmi specifických podmínek nebo dokonce jen za podmínek, které mohou existovat pouze teoreticky. Samotná existence zranitelnosti tedy nemusí nutně znamenat problém.

Obr. 1 Role zranitelnosti při útoku

Pro vyhodnocení nebezpečnosti dané zranitelnosti pro určitý systém nebo organizaci slouží analýza rizik. Vazby mezi zranitelností a ostatními součástmi pro management rizik dle ISO/IEC TR 13335 jsou ilustrovány na obrázku 2. Zranitelnost je zde nedílnou součástí všech aktiv (systémů), svou existencí zvyšuje rizika, protože může být využita některými hrozbami.

Detailnější popis analýzy rizik není náplní tohoto dokumentu, nicméně je dobré si uvědomit, že samotná existence zranitelnosti neznamená automaticky možnost úspěšného útoku. Přítomnost zranitelnosti je pouze nutná, ale nikoliv postačující podmínka pro provedení útoku.



Obr. 2 Role zranitelnosti v analýze rizik podle ISO/IEC TR 13335

3 Klasifikace zranitelností

Tato kapitola je věnována základní klasifikaci zranitelností. Čtenář tak může zjistit, s jakými typy zranitelností se lze setkat. Prakticky lze zranitelnosti klasifikovat do tří základních skupin, byť existuje celá řada způsobů jejich třídění. Základní členění je na zranitelnosti vzniklé

- nevhodným návrhem,
- nevhodnou implementací a
- nevhodným používáním.

Každou zranitelnost lze přiřadit právě do jedné z výše uvedených skupin, přičemž jeden systém může obsahovat více zranitelností různých typů. Detailnějšímu popisu jednotlivých skupin jsou věnovány následující podkapitoly.

3.1 Zranitelnosti vzniklé nevhodným návrhem

Do této kategorie spadají zranitelnosti vyplývající z nějakého nedostatku nebo opomenutí při návrhu systému (ang. design flaw, méně často též design weakness). Typickým příkladem zranitelnosti této kategorie mohou být plaintextově přenášené přístupové údaje v protokolech ftp a telnet. Problematické u tohoto druhu zranitelností je, že v době návrhu systému může být vše v pořádku a zranitelnost může vzniknout až za nějaký čas, což obvykle souvisí se změnou prostředí, ve kterém je systém provozován, a s technickým pokrokem (např. kvůli vyššímu výpočetnímu výkonu je potřeba používat silnější šifrování).

Odstranění zranitelnosti vzniklé nevhodným návrhem je nejbolestivější ze všech druhů zranitelností. Použitá technologie totiž musí být zásadně inovována

(např. ftps namísto ftp) nebo nahrazena (např. používání ssh místo telnetu). Případně může být dále provozována ve velmi omezených podmínkách (např. již zmiňovaný telnet). Nalezení vhodného řešení pro tento typ zranitelnosti obvykle přesahuje rámec jednoho dodavatele a vyžaduje širší spolupráci na mezinárodní úrovni.

3.2 Zranitelnosti vzniklé nevhodnou implementací

I když je návrh systému zcela v pořádku, lze v praxi očekávat zranitelnosti způsobené nevhodnou realizací tohoto návrhu, ať už jde o výrobu hardware nebo vytváření software.

Nejčastějším zástupcem této kategorie jsou *softwarové zranitelnosti* (ang. software vulnerabilities). Statistika udává, že na každých 1 000 řádek programového kódu se v průměru vyskytuje 5 až 15 chyb [1]. Naštěstí pouze nepatrná část z nich souvisí s bezpečností a lze je tedy označit jako zranitelnosti, ale i tak lze předpokládat, že jich existuje nezanedbatelné množství. Dočasným řešením může být vypnutí zranitelné části systému, což obvykle vede k omezení funkcionality celého systému. Trvalým řešením tohoto typu zranitelností je pouze instalace bezpečnostích záplat pro danou zranitelnost.

Méně často se lze setkat také s *hardwarovými zranitelnostmi* (ang. hardware vulnerabilities). Typicky se může jednat o zneužitelné chyby BIOSu, procesoru nebo jiného zařízení. V případě objevení takovéto zranitelnosti bývá řešení problémové, protože nezbývá než vadné zařízení celé vyměnit. Některé zranitelnosti hardware lze také odstranit úpravou jejich řídicího software (firmware).

Odstranění zranitelnosti vzniklé nevhodnou realizací je obvykle záležitostí konkrétního dodavatele systému. Organizace používající příslušný systém však také mohou, zejména v případě svobodného software, vyvíjet vlastní dočasné řešení.

3.3 Zranitelnosti vzniklé nevhodným používáním

I v případě, že by návrh systému i jeho implentace byla bezchybná, se lze setkat se zranitelnostmi, které má na svědomí špatné používání daného systému. Administrátor jej totiž nejprve musí nastavit pro specifické prostředí organizace a uživatelé systému jej pak používají. Obě tyto činnosti lze však také dělat špatně, s negativními důsledky pro bezpečnost.

Pokud administrátor nezná dostatečně spravovaný systém, jeho možnosti a prostředí v němž je provozován, může snadno vytvořit *konfigurační zranitelnost* (ang. configuration vulnerability). Typickým příkladem takového jednání je ponechání administrátorského účtu s přednastaveným heslem nebo dokonce bez hesla, ponechání nepoužívané služby v běhu nebo volný přístup ke sdíleným prostředkům. Vzhledem ke složitosti a rozsáhlosti současných systémů se

jedná o velmi běžnou záležitost. Odstranění takovéhoto druhu zranitelnosti je ve srovnání s ostatními druhy relativně jednoduché, protože je plně v možnostech organizace, pod kterou administrátor spadá. Jedinou možnou prevencí je kvalifikovaný administrátor a také co nejjednodušší systém.

Dalším kritickým místem pro bezpečnost jsou samotní uživatelé, kteří mohou i bezpečný systém používat nebezpečným způsobem, a tak vytvořit *zranitelnosti vycházející z užívání* (ang. usage vulnerabilities). Za charakteristické zranitelnosti lze považovat např. nevhodné zacházení s heslem nebo spuštění nebezpečné přílohy e-mailu. Dalším typickým příkladem je nízká odolnost uživatelů proti sociálnímu inženýrství, phishingu apod. Řešení, ale i preventivní odstranění takovýchto zranitelností spočívá ve školení uživatelů. V souvislosti s uživateli jsou občas také zmiňovány *zranitelnosti zásad bezpečnosti* (ang. policy vulnerabilites), které jsou způsobeny neexistencí nebo nedodržováním příslušných zásad. Zde je potřeba odpovídající návody a postupy vytvořit a přimět všechny zaměstnance, aby se jimi řídili.

Zranitelnostem vzniklým nevhodným používáním se lze vyhnout pouze zaměstnáním zkušených administrátorů a dostatečným vzděláním uživatelů, proto je nutné věnovat pozornost průběžným školením.

4 Životní cyklus zranitelnosti

V této kapitole jsou popsány jednotlivé fáze života zranitelnosti a představena základní terminologie spojená s touto problematikou.

Zranitelnosti *vznikají* zpravidla nechtěně, ať už jde o nevhodný návrh systému, jeho implementaci nebo nepředpokládanou změnu podmínek, ve kterých je systém provozován. Vzhledem k tomu, že zranitelnosti vznikají mimo původní plán, nikdo o nich v době jejich vzniku netuší. Takto skryté mohou v systému přebývat i velmi dlouhou dobu a někdy se na ně za celou dobu používání daného systému ani nepřijde.

Řada dodavatelů systémů, ale i různých skupin nadšenců se intenzivně zabývá hledáním zranitelností v zajímavých systémech. Část tak činí proto, aby se snížil počet chyb v systému a zvýšila se tak jeho bezpečnost, zatímco jiní hledají vhodné skulinky pro svůj útok. A tak časem může dojít k tomu, že je zranitelnost *objevena* (ang. discovery). Po tomto objevení existuje určité období, kdy je znalost zranitelnosti dostupná pouze velmi omezené skupině lidí, která ji může využívat pro své potřeby.

V průběhu času dojde také na *zveřejnění* (ang. disclosure) zranitelnosti. Zveřejněním se rozumí publikování informací o zranitelnosti důvěryhodnými zdroji dostupnými pro všechny zájemce. Na zranitelnost může být takovýto zdroj upozorněn objevitelem zranitelnosti, případně je odhalena poté co ji někdo začal zneužívat k útokům.

V souvislosti se zveřejňováním zranitelností vyvstává také otázka, je-li tato aktivita obecně přínosná, když se informace dostávají nejen ke správcům a dodavatelům systému, ale také k širšímu okruhu potenciálních útočníků. Je známým faktem, že bezpečnost založená na tajemství je velmi křehká, protože tajemství dříve či později vyjde najevo a nelze jej znovu skrýt [3]. Vhodnějším přístupem než se snažit problém utajit, je snaha o jeho vyřešení. Zveřejnění informací k tomuto řešení přispívá – dodavatel ví, že má něco opravit a nemůže se snažit problém utajit. Uživatele vědí, že si mají dávat pozor a následně instalovat příslušné záplaty. Také mohou lépe odhadovat rizika a přijímat vlastní protiopatření. Jistota, že informace mají k dispozici všichni zúčastnění, je akceptovatelnější než možnost, že se informační výhoda může přiklonit pouze k útočníkům.

V momentě kdy se potenciální útočník dozví o existenci zranitelnosti – buď ji sám objevil nebo se dozvěděl o jejím zveřejnění – započnou práce na tzv. *exploitu* (v současné době neexistuje český ekvivalent k tomuto termínu). Jde o nalezení způsobu, jak zranitelnost využít pro své potřeby. Obvykle také vznikne nějaký nástroj, kterým lze na zranitelností postižený systém provést útok. Od této chvíle je zranitelnost prakticky zneužitelná. Zejména je-li exploit také publikován ve formě jednoduchého nástroje použitelného i nekvalifikovanými útočníky.

Po svém zveřejnění se informace o zranitelnosti samozřejmě dostane také do rukou výrobce příslušného systému. Pokud jde o chybu SW vydávaného solidní organizací, je časem vydána *záplata* (ang. patch), která zranitelnost odstraní. Občas se také stává, že zranitelnost nelze odstranit a je potřeba se s její existencí smířit (např. plaintextově přenášené přístupové údaje v protokolu ftp) nebo danou technologii zásadně inovovat, případně zcela opustit. Pokud zranitelnost objeví sám dodavatel nebo jemu spřízněný jedinec, může být záplata vydána ještě před vlastním zveřejněním zranitelnosti.

Zde je na místě podotknout, že mezi zveřejněním zranitelnosti, zpřístupněním záplaty a zpřístupněním exploitu existuje velmi úzká souvislost. Pokud nastane jedna z těchto událostí, zbylé dvě na sebe nenechají dlouho čekat, protože ze záplaty lze zpravidla zpětně zjistit, co opravuje a na jejím základě vytvořit exploit a používáním exploitu dojde ke zveřejnění zranitelnosti. Statistika těchto časových souslednosti za léta 1996–2007 je pěkně popsána v článku [2].

Vydané záplaty jsou následně instalovány na zranitelné systémy. Prakticky je takových systémů vysoký počet a může trvat relativně dlouho, než se správci dozví, že záplata existuje a než bude nainstalována, protože ne každý správce může kdykoliv odstavit systém kvůli údržbě. Zranitelnost je tak málokdy zcela vymýcena ze všech systémů, protože o řadu systémů jejich správci dostatečně nepečují. Zranitelnost tedy nelze nikdy považovat za globálně zcela *eliminovanou*, ale vždy pouze ve vztahu k určitému systému.

Výše popsané klíčové momenty v životě zranitelnosti jsou ilustrovány na obrázku 3. Uvedená posloupnost uvažuje nejčastější situaci, tj. záplata je k dispozici až po zveřejnění zranitelnosti.

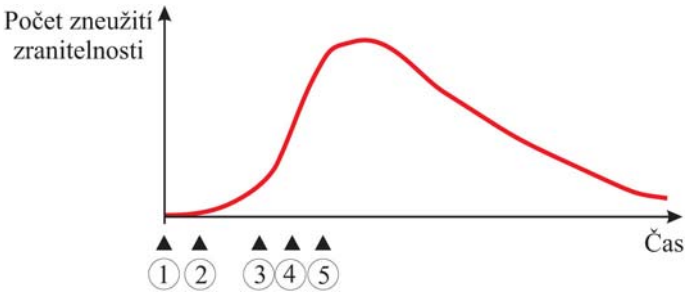
Obr. 3 Klíčové momenty v životě zranitelnosti

- Do svého objevení není zranitelnosti zneužitelná, protože o ní nikdo netuší.
- V období mezi objevením a zveřejněním může na zranitelnost reagovat – výrobou exploitu, záplaty nebo provizorního zabezpečení – pouze omezená skupina, ostatní o ní nevědí a zůstavají tak nepřipraveni.
- Teprve po zveřejnění zranitelnosti získají přístup k informacím všichni a mohou je využít. Situace se oproti předchozímu stavu mírně zlepšila, administrátor už ví, že jeho systému hrozí nebezpečí a může se pokusit vlastními silami vhodně reagovat.
- V momentě, kdy dodavatel systému vydá příslušnou záplatu, je situace mnohem příznivější. Je totiž znám způsob jak zranitelnost ze systému rozumně odstranit. Informovanost je na svém vrcholu – každý může zjistit, že zranitelnost existuje a jak ji odstranit.
- Zranitelnost je na daných systémech eliminována až po instalaci příslušné záplaty.
- Co se týká exploitu využívajícího danou zranitelnost, tak je pro určitý systém použitelný od svého vzniku až do doby, než je na systém instalována příslušná záplata.

Na obrázku 4 je znázorněn typický průběh počtu zneužívání zranitelnosti v čase. Po svém objevení a následně i zveřejnění je zranitelnost nejprve zneužívána málo, protože většina lidí není schopna využít informaci o zranitelnosti sama a její masivní zneužívání tak započne až poté, co je exploit dostupný ve snadno použitelné formě (skript, aplikace). Po prudkém nárůstu počtu zneužití se jejich počet začne pomalu snižovat postupnou instalací záplat na jednotlivé systémy. Jak už bylo řečeno dříve, může být tento proces oddálen nebo vůbec nespuštěn z různých důvodů, proto je snižování velmi pozvolné.

5 Práce s informacemi o zranitelnostech

Následující odstavce popisují, kde najít informace o zveřejňovaných zranitelnostech a jak se v nich orientovat. Také jsou zmíněny související standardy.



Obr. 4 Zneužívání zranitelnosti v čase: Objevení (1), zveřejnění zranitelnosti (2), zveřejnění exploitu (3), vytvoření záplaty (4), zahájení instalace záplat (5)

5.1 Informační zdroje

Je v zájmu každého administrátora systému, aby věděl jaké zranitelnosti týkající se jeho systémů byly zveřejněny. Pro včasnou informovanost je tedy vhodné sledovat informační zdroje zabývající se zranitelnostmi. Zpravidla se jedná o

- stránky příslušného CSIRT (Computer Security Incident Response Team),
- stránky nebo mail-list dodavatele konkrétního systému,
- stránky nebo mail-list specializovaného serveru – např. Bugtraq [4] a
- články v odborných publikacích – zde však může být určité časové zpoždění mezi zveřejněním zranitelnosti a vydáním publikace.

Sledováním výše uvedených zdrojů lze zajistit včasné zachycení konkrétních zranitelností. Nicméně je užitečné sledovat i zdroje zabývající se také zobecněným pohledem na zranitelnosti a bezpečnostní doporučení jako jsou

- www.root.cz,
- www.securityfocus.com,
- www.isc.sans.org,
- www.schneier.com (zejména sekce „Crypto-gram Newsletter“ a „Cryptography and Computer Security Resources“) nebo
- www.securiteam.com.

Pro pohodlnější a přehlednější sledování většího množství informačních zdrojů je vhodné využívat RSS kanály (většina výše uvedených zdrojů jej nabízí). Informace ze všech zdrojů tak mohou být soustředěny na jedno místo např. webovou stránku a není nutné pravidelně procházet stránky jednotivých zdrojů.

5.2 Identifikace chyby

Vzhledem k množství používaných systémů a jejich rozsáhlosti jistě nikoho nepřekvapí, že počet existujících zranitelností je velmi vysoký. Některé jsou sice časem opraveny, ale vývojáři hbitě přidávají další a další. I když je objevena a posléze zveřejněna pouze část z těchto zranitelnosti, pořád se jedná se o vysoké číslo. Související informace jsou rozprostřeny po celé řadě serverů věnujícím se bezpečnostní tématice a sehnat všechny materiály ke konkrétní zranitelnosti je bez její jednoznačné identifikace prakticky nemožné. Proto jsou jednotlivým zranitelnostem přidělovány identifikátory ulehčující vyhledávání a agregaci informací.

V roce 1999 bylo spuštěno popisování zranitelností *identifikátorem CVE* (Common Vulnerabilities and Exposures) a v současné době jde o nejrozšířenější veřejně dostupný identifikátor zranitelností, který je všeobecně akceptován. Každá takto katalogizovaná zranitelnost obsahuje

- identikátor CVE – např. CVE-2008-1447,
- krátký popis zranitelnosti – jakých systémů se týká, co může způsobit za problém, apod.,
- vybrané odkazy – slouží pro upřesnění zranitelnosti, nejde o úplný výčet,
- status – buď kandidát na zápis (candidate) nebo zapsaná položka (entry), závisí na tom zda byl editory schválen a
- datum přidělení identifikátoru.

Do kompletního katalogu zranitelností s identifikátorem CVE lze nahlédnout na jeho domovské stránce [7], případně na jiných stránkách, které obsahují jeho kopii, např. [8]. Rozumné vyhledávání zranitelností pro jednotlivé systémy poskytuje také např. [9].

5.3 Závažnost zranitelnosti

Pro každou zveřejněnou zranitelnost je potřeba vyhodnotit její závažnost, tj. zjistit za jakých podmínek je prakticky zneužitelná a k čemu všemu může dojít. Aby stejnou práci nemuseli dělat všichni administrátoři systémů, existuje *standard pro ohodnocování závažnosti* zranitelností označovaný zkratkou *CVSS* (Common Vulnerability Scoring System) [5].

Ze způsobu jakým je závažnost určována, lze získat komplexní pohled na fungování zranitelnosti a jejich následné vyhodnocování. Nejprve jsou *základní metrikou* popsány vlastnosti zranitelnosti, které zůstávají po celou dobu existence zranitelnosti neměnné. Skládá se z vyhodnocení

- vektoru přístupu = odkud je zranitelnost dostupná (lokální uživatel, lokální síť nebo vzdáleně),
- složitosti útoku = zda existují další omezení (např. časová, konfigurační, apod),
- způsobu autentizace = zda-li je nutné se autentizovat (případně kolikrát),
- dopadu na důvěrnost = jakým způsobem bude narušena důvěrnost cílového systému (vůbec, částečně, úplně),
- dopadu na integritu = jakým způsobem bude narušena integrita cílového systému (vůbec, částečně, úplně) a
- dopadu na dostupnost = jakým způsobem bude omezena dostupnost cílového systému (vůbec, částečně nebo úplné vypnutí).

Dále je definována *časově proměnná metrika*, vyvíjející se během života zranitelnosti. Bere v potaz

- zneužitelnost = aktuální možnosti pro zneužití zranitelnosti (není známo jak zranitelnost zneužít, je znám princip nebo již existuje exploit, případně není exploit potřeba),
- eliminovatelnost = jakým způsobem lze zranitelnost odstranit (vůbec, neoficiální záplata, oficiální záplata, komplexní řešení) a
- důvěryhodnost zpráv o zranitelnosti = jakým způsobem je o zranitelnosti a možnostech její eliminace informováno (neověřený zdroj, neoficiální zdroje, dodavatel systému).

Jako poslední je uvažována *metrika prostředí*, která se snaží popsat nebezpečnost zranitelnosti pro prostředí, ve kterém se zranitelné systémy běžně vyskytují. Bere ohled na

- možnosti souběžného poškození = jaký vliv na okolní systémy může daná zranitelnost mít,
- rozložení cílů = kolik systémů s danou zranitelností existuje a
- důsledek = jak vážné mohou být dopady.

Na základě výše uvedených kritérií je pak, podle určitého matematického vzorce, ke každé zranitelnosti přiřazena závažnost reprezentovaná číslem od 0 do 10. Čím vyšší hodnota tím je zranitelnost závažnější a měla by jí být věnována větší pozornost. Databázi zranitelností i s odhodnocenou závažností lze nalézt např. na [10].

5.4 Některé další standardy

Jednotlivé týmy a organizace, které se zabývající hledáním, zveřejňováním, katalogizováním a všeobecně problematikou zranitelností, si potřebují mezi sebou vyměňovat data. Proto existují některé další standardy.

CAIF (Common Announcement Interchange Format) – standard formátu pro výměnu informací o zranitelnostech, detailní informace na stránce [11].

AVDL (Application Vulnerability Description Language) – standard OASIS pro formát popisu zranitelností aplikací.

WAS (Web Application Security) – standard OASIS pro formát šíření informací o zranitelnostech.

EISPP (European Information Security Promotion Programme) – projekt EU pro vytvoření sdílené DB o zranitelnostech.

6 Zranitelnosti a CSIRT

Obsahem této kapitoly je popis funkce bezpečnostního týmu CSIRT v souvislosti se zranitelnostmi. V první řadě CSIRT *interaguje se svými uživateli* (zákazníky), pomáhá jim s informovaností ohledně zranitelností a vysvětluje praktické důsledky. Případně může také dojít k vlastnímu *zkoumání zranitelnosti* nebo dokonce ke *spolupráci s ostatními CSIRT* při tomto výzkumu. V praxi je zpravidla k vidění pouze prvně zmiňovaná funkce.

6.1 Šíření informací mezi zákazníky

Základní funkcí CSIRT týmu v souvislosti se zranitelnostmi je *šíření příslušných informací* (ang. distribution). Tato služba předpokládá, že cíloví uživatelé (zákazníci) pouze nemají čas sami sledovat všechny informační zdroje, případně nejsou schopni komunikovat v cizím jazyce, ale po předání příslušné informace jsou schopni správné reakce. Pro šíření těchto informací je vhodnou formou webová stránka bezpečnostního týmu nebo mailing list pro příslušné správce. Oboje by mělo být vhodně zajištěno (SSL, digitální podpis), aby se zamezilo podvrhnutým radám. Každá zpráva o zranitelnosti by měla obsahovat

- referenční číslo (pro lepší orientaci ve zprávách),
- informaci pro koho je zpráva určena (manažer, správce, uživatel),
- typ zprávy (předběžné upozornění, průběžná zpráva, koncové řešení),
- čeho se zranitelnost týká (SW, služba, uživatel, apod),
- jaké mohou být důsledky (výpadek služby, narušení integrity, apod.),

- jaká je šance, že zranitelnost někdo využije (úzce souvisí s CVSS),
- jak zjistit, že zranitelnost je zrovna u mne (aplikace, skript nebo jiný test),
- jak problém vyřešit (existence záplaty, neoficiální řešení, nelze odstranit) a
- jaké mohou být dopady řešení (co se nápravou zranitelnosti může změnit).

Veškeré takto šířené zprávy je dobré archivovat zejména pro studijní a statistické účely.

6.2 Vysvětlování

Svět není ideální, takže nemalé množství uživatelů (zákazníků) není schopno samostatného zpracování předávané informace o zranitelnosti. Jednak nemusí rozumět odbornému textu a jednak nemusí správně vyhodnodit důsledky. Proto řada bezpečnostních týmu CSIRT k šířeným informacím poskytuje také *vysvětlení* (ang. interpretation). Obsahem je shrnutí pro laickou veřejnost a přesný návod co je potřeba udělat, např.

- spustit záplatu, včetně odkazu ke stažení,
- změnit konfiguraci, nejlépe se vzorovým konfiguračním souborem a obrázkovým návodem,
- neprovádět určité aktivity, např. nečíst e-maily s hlavičkou „Nigérijská národní loterie“ apod.

Šíření informací spolu s jejich vysvětlováním se v žádné případě neobejde bez spolupráce s HelpDeskem, kam budou uživatelé volat a klást zvídavé otázky. Je tedy vhodné každou zprávu vysvětlit také operátorům HelpDesku, aby tazatelům odpovídali konzistentně.

6.3 Vlastní výzkum a spolupráce s ostatními

Nechce-li daný CSIRT pouze čekat na informace o zajímavé zranitelnosti, může se také sám pustit do jejího *zkoumání* (ang. investigation). Cílem je zejména lepší porozumění danému problému a hledání dočasného řešení do doby, než bude k dispozici oficiální záplata.

Jsou-li ambice týmu ještě vyšší, dochází k *součinnosti* (ang. coordination) s ostatními CSIRT týmy zabývajícími se stejnou zranitelností. Výsledkem tohoto snažení pak obvykle bývá návrh dočasného nebo dokonce úplného řešení dané zranitelnosti.

7 Popis vybraných zranitelností

Tato kapitola představuje vybrané zranitelnosti a principy jejich fungování. Příklady jsou věnovány zranitelnostem vzniklým nevhodnou implementací, se kterými se lze velmi často setkat.

7.1 Buffer overflow

Tento typ zranitelností je znám již velmi dlouho (přes 30 let), nicméně stále jde o nejčastěji zneužívanou zranitelnost. Důvodem je fakt, že je lze nalézt téměř všude a útočníkovi umožňují relativně pohodlně spustit libovolný kód.

V paměti počítače jsou uloženy dva druhy informací – data, se kterými aplikace pracuje a vlastní kód aplikace. Buffer overflow využívá skutečnosti, že při spuštění aplikace nejsou tyto oblasti nijak zřetelně odděleny a je možné zařídit, aby data byla interpretována jako kód. Prakticky je toho dosahováno tak, že vstupní data mají větší objem než oblast, do které jsou ukládána. Dojde tak k přepsání obsahu paměti sousedící s cílovou oblastí. Vzhledem k tomu, že se tam může vyskytovat také kód aplikace, lze činnost dané aplikace úplně změnit. Potenciální útočník tedy musí přepsat obsah paměti vhodnými daty – v opačném případě nedosáhne svého původního cíle a zpravidla dochází k pádu aplikace.

Skutečnost, že je umožněno zapsat data také do paměti, kde nemají co dělat, je způsobena chybou programátora. Nejčastěji se jedná o

- nehlídané hraniční indexy polí,
- špatně implementované indexování, nevhodné používání jinak bezpečných funkcí (např. `strncpy()` apod.) a
- problematické zacházení s číselnými typy.

Ochrana proti buffer overflow není nikterak jednoduchá a neexistuje univerzální řešení. Eliminace této zranitelnosti může být v ideálním případě dosaženo bezpečným programováním, prakticky pak pomáhá rozdělení paměti na datovou a aplikační část (např. pomocí technologie NX pro novější procesory), ochrana kritických dat (např. návratové adresy) v zásobníku (stack) proti přepsání nebo náhodné umístění oblastí zásobníku a hromady (heap), které znesnadňuje sestavení vhodného řetězce pro zneužití dané aplikace.

Další užitečné informace o buffer overflow lze nalézt např. v [12, 13].

7.2 Format string vulnerabilities

Možnost zneužití formátování řetězce byla objevena v druhé polovině roku 2000 a od té doby patří, spolu s buffer overflow, k nejčastěji zneužívaným zranitelnos-

tem. Většina těchto zranitelností se vyskytuje v programech psaných v jazyce C/C++, ale lze se s nimi také setkat i jinde.

Má-li být řetězec vypsán na výstupní zařízení, je pro tento účel použita příslušná funkcí (např. `printf()`). Aby mohly být vypsány také různé proměnné, mohou být součástí řetězce také tzv. řídicí sekvence (např. `%d`, `%20s`, `%n`, apod.). Každé řídicí sekvenci by v programovém kódu měl odpovídat jeden parametr funkce, nicméně počet není samotnou funkcí nijak kontrolován a automaticky je vždy brána proměnná na vrcholu zásobníku. Prakticky se tedy může stát, že počet argumentů funkce a počet řídicích sekvencí je různý. Formátovací funkce funguje tak, že postupně prochází řetezec a vypisuje jej do zvoleného výstupu. Narazí-li na řídicí sekvenci, sáhne do zásobníku a příslušná data vypíše. Prakticky je tedy pro každou řídicí sekvenci vypsána část paměti, přičemž velikost a způsob zobrazení jejího obsahu je definován touto sekvencí. Nyní je na místě podotknout, že v zásobníku je také uložen vlastní formátovací řetězec a tedy podobně jako u buffer overflow nejsou data a instrukce k jejich zpracování (zde „pouze“ formátování řetězce) zřetelně oddělena. Pokud je útočníkovi umožněno ovlivňovat způsob formátování vypisovaného řetezce, tj. je možné zadat řídicí sekvenci pro formátovací funkci, pak jde o format string vulnerability. Typickým příkladem je může být např. zobrazení hlášky `Neznámý příkaz: <zadaný příkaz>`, kde stačí vhodně zadat onen neznámý příkaz. Tento typ zranitelnosti lze zneužít k několika typům útoků:

- *ukončení (pád) služby*, což může útočníkovi posloužit, např. zajímavou službu přinutí k výpisu paměti (core dump) nebo službu odstaví, aby mohl podvrhnout vlastní verzi. Přinutit nějakou aplikaci k pádu je relativně jednoduché, stačí zadat více řídicích sekvencí než je počet parametrů příslušné funkce a dříve či později dojde k pokusu o vypsání obsahu paměti, ke které proces nemá přístup a následnému pádu.

- *zobrazení obsahu paměti*, ve které se mohou nacházet zajímavé údaje. Tento způsob zneužití již vyžaduje hlubší analýzu dané služby, protože může snadno dosáhnout výše uvedeného důsledku (pád aplikace). Při použití pokročilých technik není nutné se omezit pouze na paměť zásobníku.

- *přepsání obsahu paměti*, což je snem každého útočníka, protože může změnit návratové adresy funkcí a vložit vlastní kód. Pro tento účel je používána řídicí sekvence `%n`, která do zvolené proměnné (parametr funkce) zapíše počet aktuálně vypsaných bytů. Vhodně zadaný řetězec pak může způsobit přepsání paměti zvolenou hodnotou.

Detaily o technice zneužívání format string vulnerabilites jsou přehledně popsány v [6]. Jedinou možnou ochranou je čisté zacházení s řídicími řetězci formátovacích funkcí.

7.3 SQL injection

S dotazy formulovanými v SQL (Structured Query Language) se lze v současnosti setkat takřka u všech databází. Způsob vytváření strukturovaných dotazů totiž umožňuje jednoduchým a pružným způsobem získat data, která jsou potřeba [14].

Současně je však SQL také náchylné ke zneužití pomocí tzv. SQL injections. Princip této zranitelnosti spočívá v umístění uživatelem zadávaného řetězce do SQL dotazu. Typickým případem je následující dotaz, který zprostředuje údaje o uživateli dle zadaného jména a hesla

```
SELECT * FROM users WHERE user='$name' AND password='$psswd'.
```

Pokud útočník zadá do řetězců `$name` nebo `$passwd` vhodný řetězec, může zcela změnit původně zamýšlený výsledek dotazu. Prakticky může provést útok z následujících kategorií [15]:

- *SQL manipulation* – nejčastější a nejdéle známý způsob SQL injections. Základní myšlenkou je modifikace daného dotazu tak, aby umožnil přístup k více záznamům, než má daný uživatel dovoleno. Například zadáním uživatelského jména i hesla ve tvaru `' OR 1=1 --` lze docílit, že dotazu budou vyhovovat všechny záznamy. S použitím dalších direktiv jako `UNION` lze docílit i mnohem zajímavějších výsledků.

- *Code Injection* – útoky z této kategorie se neomezují pouze na změnu parametrů zneužívaného dotazu, ale přidávají za něj další, uživatelem vytvořený, dotaz. Např. smazání tabulky lze provést zadáním hesla ve tvaru `'; DROP TABLE users;`. U některých databází je násobné vykonávání dotazů v rámci jednoho požadavku zákázáno, takže jsou proti této verzi útoku imunní.

- *Function Call Injection* – řada databází umožňuje v rámci dotazu používat různé standardní funkce jako jsou převody mezi formáty, změna velikosti písmen, ale také posílání http požadavků apod. Některé databáze podporují také TSQL (Transact-SQL), takže se v rámci SQL dotazu muže vyskytnout např. `shutdown with no wait`. Obojí může zkušený útočník může využít pro své potřeby

- *Buffer overflow* – funkce zmíněné v předchozím bodě mohou obsahovat další zranitelnosti a vhodným zadáním parametrů lze dosáhnout přetečení zásobníku. Následky byly naznačeny již v kapitole 7.1.

Pokud útočník zná strukturu jednotlivých tabulek a jejich vzájemné vazby, má výrazně ulehčenou práci. Nicméně i databáze, jejichž struktura není veřejně známa, nejsou vůči této zranitelnosti imunní. Neohlídaná chybová hlášení mohou často prozradit potřebné informace, nehledě k tomu, že bezpečnost založená na tajemství nemá dlouhého trvání.

Ochrana proti tomuto typu zranitelností spočívá v pečlivém ošetření uživatelských vstupů, omezení uživatelem spouštěných funkcí a omezení obsahu chybových hlášení [15].

8 Závěr

Po přečtení tohoto dokumentu by čtenář měl vědět co si představit pod pojmem zranitelnost (vulnerability) a s jakými druhy zranitelností se může setkat. Dále byla věnována pozornost životnímu cyklu zranitelnosti od jejího vzniku až po její eliminaci, přičemž bylo popsáno, kde se dají informace o vzniklých zranitelnostech najít a jak s nimi nakládat. Následně byl objasněn vztah CSIRT k fenoménu zranitelností. Na závěr pak byly podrobněji popsány vybrané zranitelnosti.

Reference

[1] Convery, S.: *Network Security Architectures*, Cisco Press, 2004, ISBN 158705115X.

[2] Frei, S., May, M., Fiedler, U., Plattner, B.: *Large-Scale Vulnerability Analysis*, Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense, Pisa, Italy, 2006, pp. 131–138, ISBN 1-59593-571-1.

[3] Scheier, B.: *The Nonsecurity of Secrecy*, Communications of the ACM, October 2004, vol 47, issue 10, pp. 120, ISSN 0001-0782.

[4] http://www.securityfocus.com, odkaz Bugtraq.

[5] http://www.first.org/cvss/intro/index.html

[6] http://julianor.tripod.com/bc/formatstring-1.2.pdf

[7] http://cve.mitre.org/cve/index.html

[8] http://www.securiteam.com/cves/

[9] http://www.securityfocus.com/vulnerabilities

[10] http://nvd.nist.gov/nvd.cfm

[11] http://www.caif.info/

[12] http://it.toolbox.com/blogs/adventuresinsecurity/stop-buffer-overflow-attacks-against-unpatched-vulnerabilities-8730

[13] http://archiv.computerworld.cz/cwarchiv.nsf/clanky/AE047F81B57DDDDFC12571170053A1A4?OpenDocument

[14] Harper, M,: *SQL Injection Attacks – Are You Safe?*, http://www.sitepoint.com/

[15] Integrigy: *Introduction to SQL Injection Attacks for Oracle Developers*, http://www.net-security.org/

Cloud computing

Dalibor Kačmář

E-mail: dalibor.kacmar@microsoft.com

Abstrakt

Cloud computing je v současné době téma, které zaměstnává celou řadu IT odborníků a vývojářů. Microsoft ohlásil podporu této formy realizace IT na podzim roku 2008 pod názvem Platforma služeb Azure. V prezentace bude proto věnovaná jak pohledu na konkrétní přínosy využití cloud computingu, ale i konkrétním ukázkám dostupných služeb, které může každá firma i koncový uživatel dnes získat. Podíváme se i na integraci cloud platformy s ostatními online službami a popíšeme si na praktických příkladech jak mohou firmy již dnes Microsoft Azure služby využít.

Autor příspěvěk nedodal.

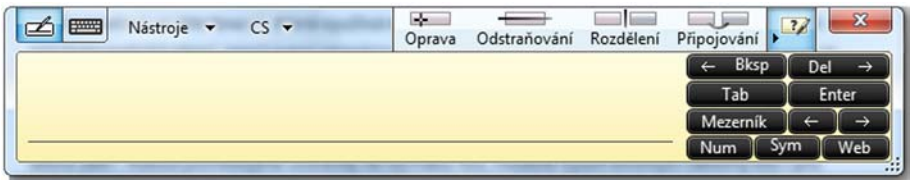
Rozpoznávání rukou psaného textu ve Windows 7

Štěpán Bechynský

E-mail: stepan.bechynsky@microsoft.com

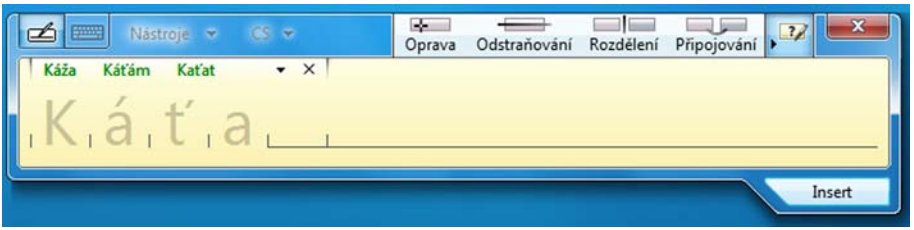
Podpora pro rozpoznávání rukou psaného textu se v operačním systému Windows objevila poprvé v listopadu 2002 s příchodem Tablet PC a Windows XP Tablet PC Edition. Tablet PC jsou typicky notebooky s „dotykovou obrazovkou“, která funguje na podobném principu jako tablet, takže většinou potřebujete speciální pero. Díky možnosti otočit víko notebooku obrazovkou nahoru jsou Tablet PC uzpůsobené pro psaní i ve chvíli, kdy musíte notebook držet v ruce a nemáte možnost si ho položit. Pro nás podstatná změna ve Windows 7 je podpora češtiny. Během přípravy podpory pro rozpoznávání češtiny odevzdalo svůj rukopis několik set lidí z řad zaměstnanců české pobočky společnosti Microsoft, jejich příbuzných a známých. Cílem bylo získat co největší spektrum rukopisů od malých dětí, které píší „krasopisem“ až po „vypsané“ ruce starších ročníků. Poslední sběr rukopisů proběhl v únoru 2009, kdy se nasbíralo kolem padesátky dalších vzorků.

Pokud máte nainstalovánu podporu pro Tablet PC, máte k dispozici panel pro zápis textu rukou, který se aktivuje při přiblížení pera k obrazovce.



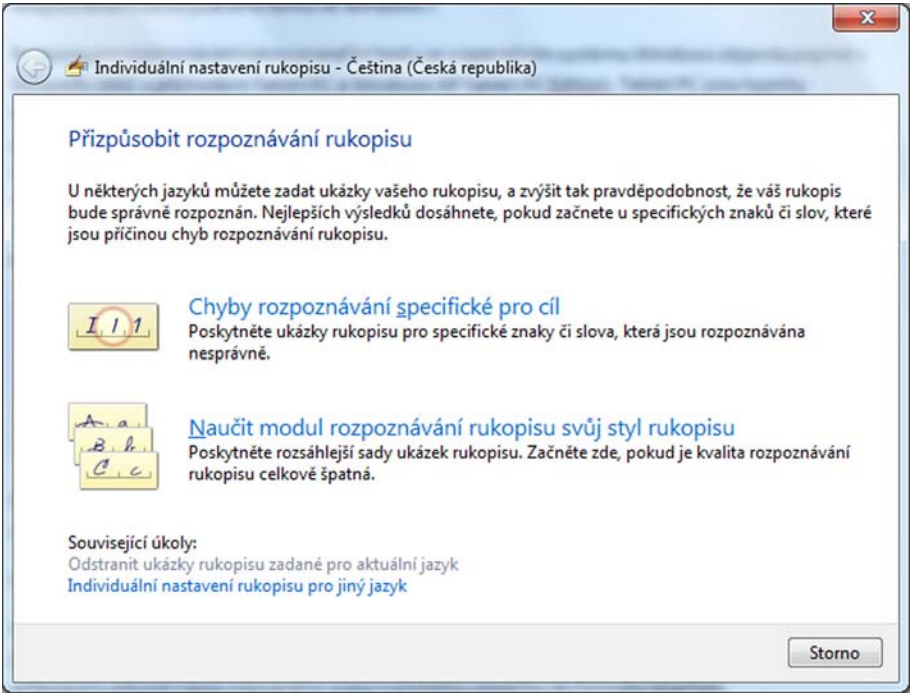
Obr. 1

Pokud je rozpoznání textu chybné, máte možnost text opravit a tím se celý systém učí. Jak vidíte na snímku obrazovky, během oprav jsou vám nabízena další potencionálně správná slova. Systém tedy nepracuje jen s analýzou tahů perem, ale také spolupracuje se slovníkem. Tím se úspěšnost rozpoznání výrazně zvyšuje.



Obr. 2

Druhá možnost je hned na začátku používání Tablet PC spustit aplikaci, kde opíšete 50 předpřipravených vět a tím systém naučíte lépe rozpoznat váš rukopis.

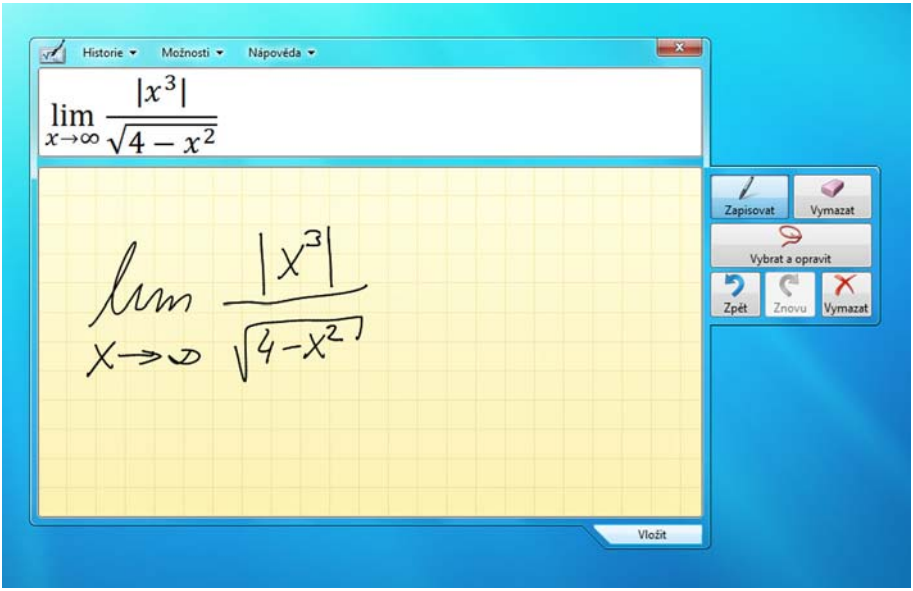


Obr. 3

Panel pro matematický zápis

Další z novinek ve Windows 7, která využívá možností rozpoznávání rukou psaného textu je „Panel pro matematický zápis“, který ocení zejména studenti a vyučující technických oborů. Tato aplikace je určena pro převod rukou napsaného matematického vzorečku do formátu MathML (http://www.w3.org/Math/). Takto vytvořený vzoreček aplikace uloží do schránky, odkud si pak můžete vzoreček vložit do cílové aplikace, která musí podporovat formát MathML, např. Microsoft Word 2007. Pokud potřebujete vzorečky ve formátu TEX, můžete využít existující šablony XSLT pro transformaci, které najdete na internetové adrese http://www.raleigh.ru/MathML/mmltex/index.php?lang=en. K převodu formátu MathML do formátu, který vyžaduje Microsoft Word 2007, se ostatně také používá XSLT. Tyto šablony najdete v adresáři, kde je nainstalován Microsoft Word 2007, v souborech MML2OMML.XSL a OMML2MML.XSL.

Panel pro matematický zápis můžete také vkládat přímo do vlastní aplikace jako COM objekt.



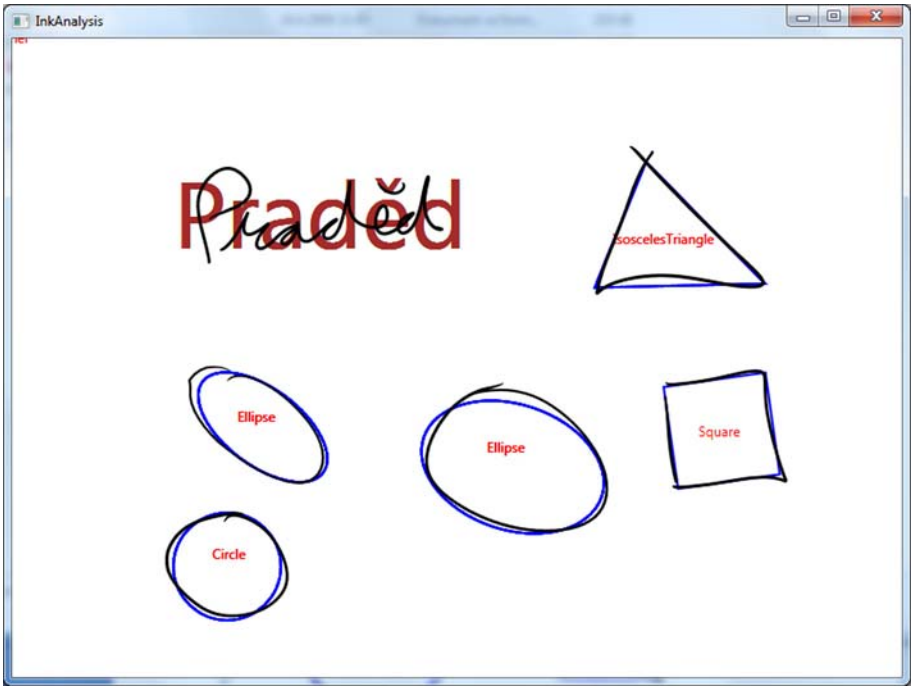
Obr. 4

InkCanvas a InkAnalyzer

InkCanvas je objekt z .NET Framework 3.0 a jedná se v podstatě, jak název objektu napovídá, o kreslící plochu. Jednotlivé tahy, které do InkCanvas nakreslíte, můžete pak analyzovat pomocí objektů ze jmenného prostoru System.Windows.Ink, který je součástí Tablet PC SDK.

```
InkAnalyzer ia = new InkAnalyzer();
ia.AddStrokes(inkText.Strokes); // object inkText je datov0ho typu
InkCanvas
ia.SetStrokesLanguageId(inkText.Strokes, 1033);
ia.Analyze();
string s˜= ia.GetRecognizedString();
```

Podobným způsobem můžete „odhadovat“ nakreslené tvary (kruh, čtverec, trojúhelník, . . .).



Obr. 5

Windows 7

Karel Florian

Nový klientský operační systém společnosti Microsoft přináší mnoho zajímavých novinek. Ať už jde o vylepšení ovládání, grafického rozhraní či novinek skrytých uživateli. Tedy změny v jádru OS, zabezpečení, možnosti správy atd.

Stávající verze operačního systému, Windows Vista, nastolila směr, kterým se nadále prostředí a schopnosti operačního systému od společnosti Microsoft budou ubírat. Hlavní zaměření funkcionality tedy zůstalo stejné:

- snadná dostupnost informací a přístup odkudkoli
- vyšší efektivita práce
- ergonomie a přímočaré ovládání
- bezpečnost

Pokud bychom rozdělili hlavní funkční změny a vylepšení do dvou základních kategorií, dostali bychom zhruba následující seznam.

Koncový a domácí uživatel:

- **Zcela nový taskbar a tzv. jump-listy**, které zjednodušují orientaci ve spuštěných programech a pomáhají uživateli efektivnější manipulovat s procesy a oblíbenými programy.
- **Vylepšené vyhledávání a tzv. federation search**. Obojí pomáhá uživateli nalézt požadované informace jak na daném počítači, tak i v internetu, popř. oblíbených a definovaných serverech.
- **Windows Explorer a knihovny.** Snazší organizace složek do tzv. knihoven, které mohou sdružovat jednotlivé vybrané adresáře a uživatel tak najde vše na jednom místě.
- **Homegroup.** Tedy zjednodušené a velmi rychle nastavitelné sdílení informací, hudby, filmů v rámci domácí sítě během několika málo kliknutí myší.

- **Zařízení a tiskárny, ovladače, streamování médií.** To vše na jednom místě, ve vylepšených ovládacích panelech a zjednodušeným průvodcům konfigurace.
- **Internet Explorer 8.** Zcela nový prohlížeč s mnoha systémovými a klíčovými vylepšeními poskytne bezpečnější, rychlejší a pohodlnější cestu k informacím na internetu.

IT odborníci/korporátní sféra:

- Nasazení pomocí systému image souborů, jejich správa a konfigurovatelnost.
- Mnohem širší možnosti v rámci skupinových politik (zvláště ve spojení s Windows Serverem 2008 R2) a také skupinových preferencí pro detailní nastavení klientské stanice.
- Zabezpečení systému za pomoci vylepšeného a konfigurovatelného User Account Control (UAC).
- Nový Internet Explorer 8 s možností nastavení za pomoci skupinových politik, popř. vytvoření vlastní konfigurace IE pro koncové uživatele.
- Direct Access, neboli služba, která umožní uživateli připojit se do práce či kanceláře bez použití VPN klienta či softwaru třetí strany, přímo za pomoci nativního Windows prostředí, samozřejmě při dodržení vysoké bezpečnosti, šifrování a všech autentizačních náležitostí.
- **Problem Steps Recorder (PSR)** je novou funkcí, která umožňuje uživateli „nahrát“ problémové chování počítače a odeslat jej správci během pár vteřin k vyřešení.

Funkcí a vylepšení je samozřejmě mnohem více a mohli bychom ve výčtu pokračovat na dalších X stranách tohoto článku. Nicméně pro utvoření alespoň základního přehledu toho, co vše od Windows 7 můžeme očekávat to postačuje.

Pokud přičteme dramaticky upravený kernel systému s mnohem vyšším výkonem, rozdělení jeho vrstev do několika úrovní kvůli zabezpečení procesů a řízení jejich logiky, spravovatelnost za pomoci Powershell V2 a podporu všech virtualizačních technologií (MED-V, APP-V), terminálových služeb, Application Locker a spoustu, spoustu dalšího, dostaneme obrázek vyspělého, výkonného, bezpečného a velmi schopného operačního systému, který pokračuje ve směru nastoleném Windows Vista.

Společně ve spojení se serverových operačním systémem Windows Server 2008 R2 budou Windows 7 tvořit velmi silnou a schopnou dvojici, která uspokojí jak náročné korporátní uživatele a administrátory, tak ještě náročnější uživatele domácí.

Agile project planning (Does your project need Gantt chart)

Václav Pergl

E-mail: vpergl@kerio.com

Motto:
"These days we do not program software module by module; we program software feature by feature."
Mařenka Poppendiecková

Introduction

Planning is important, even for agile projects and this article introduces agile planning practices and describes how to use it in multi-team project.

In agile methods, loading a team with work is done through iteration planning. Due to the shortness of the iteration (typically two weeks) a *plan* reduces in importance and *planning* gains in importance. For small projects it may be sufficient to plan only a single iteration at a time. The experienced disadvantage of iteration planning when applied to projects that run for more then a few iterations or with multiple teams is that the view of the longer term implications of iteration activities can be lost. In other words: the view of 'the whole' is lost.

A solution is to add planning levels to incorporate the current view of 'the whole'. In plan driven and waterfall methodologies this problem is overcome through a large upfront design, aiming to predict accurately how much work is involved in each project task. This leads to a large investment early in the project, when it is by no means certain that the designed functionality is actually the functionality desired by the product owner. An approach with multiple levels of planning has to avoid the reintroduction of the big design up front.

Planning activities for large-scale development efforts should rely on five levels:

- Product Vision
- Product Roadmap
- Release Plan

- Sprint Plan
- Daily Commitment

Each of the five levels of planning addresses the fundamental planning principles: priorities, estimates and commitments.

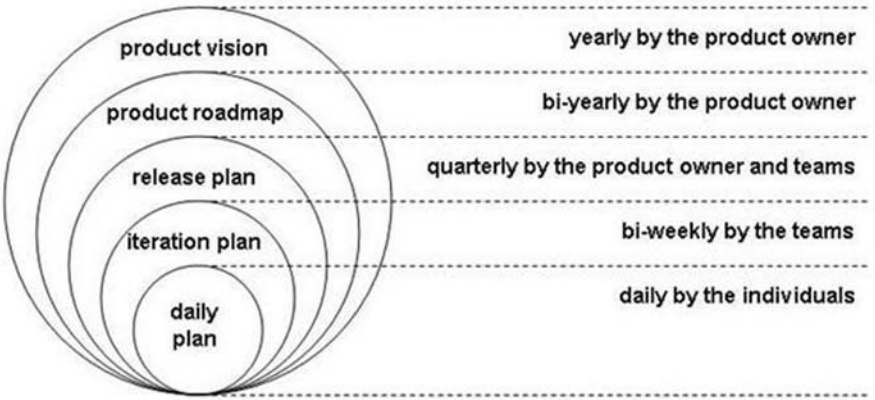


Figure 1

1 Strategic Level

1.1 Strategic Product Vision

The Product Vision is designed to present the big picture of the product. The vision defines the mission and the boundaries within which we will work to achieve the desired results. The release goal should be directly traceable to a corporate strategic objective.

Possible structures for a visioning exercise are to create an elevator statement or a product vision box. The principle of both exercises is to create a statement that describes the future in terms of desired product features, target customers and key differentiators from previous or competitive products. The following structure of the elevator statement is suggested:

"For (target customer) who (statement of the need) the (product name) is a (product category) that (product key benefit, compelling reason to buy). Unlike (primary competitive alternative), our product (final statement of primary differentiation)."

Product Vision describes a desired state that is one year or more in the future. Further planning activities will detail the vision, and may divert from the vision because the future will bring us a changed perspective on the market, the product and the required efforts to make the vision reality.

2 Product Level

2.1 Product backlog

In anticipation of the next planning stage a list of desired features needs to be built — the product backlog. In its simplest form, such a backlog is a table (spreadsheet) of product requirements, briefly described so a development team can provide estimates for the realization of each feature. Most importantly, the list has to be prioritized. The success of a project depends on the early delivery of the highest priority features. Since the success of a project is measured in business terms, the prioritization of the feature list is the responsibility of the business.

Interaction with the development teams is required. Without a discussion of the features it will be hard for the development team to produce estimates that have an acceptable inaccuracy. Characteristics of a product backlog include:

- One product backlog for all teams (see the whole)
- Large to very large features (themes)
- Feature priority based on business priorities (discovered through market research)
- Technology features (sometimes called non-functional features) are limited to those that have a direct impact on the success of the product in the market.

Typical backlog includes the following mandatory fields:

ID	a unique identification, just an auto-incremented number
Name	a short, descriptive name of the story
Priority	the priority for this story (1 = MAX, ..., 5 = MIN). or labels (acronym MoSCoW): • Must, • Should, • Could, • Wont.

Complexity	• **Size category.** The simplest way to rate features is by *size category*: small, medium, large, extra-large ("T-shirt size") or serial numbers (1 = MIN, ..., 5 = MAX). • **Story points.** Non-dimensional units of complexity that are assigned by estimators to each feature. • **Ideal man-days.** Very good understanding of the feature and the team's abilities are required. This granularity of estimation also risks misleading that people can be traded for features and time (classic fallacy of the "mythical man-month".) Just use them with care.

Auxiliary fields can be:

Acceptance test description	a high-level description of how this story will be demonstrated at the sprint demo. If you practice test-driven development this description can be used as pseudo-code for your acceptance test code.
Notes	any other info, clarifications, references to other sources of info, etc.

2.2 Impact Analysis Form

Impact Analysis Form contains assessment of effort, risks, constraints and dependencies of features/themes or requested changes. Is used for assessment of "top ten" Product Backlog items (feasibility, human resource allocation, third party dependencies, risks or budget planning).

2.3 Product Roadmap

The creation of the roadmap is largely driven by the product owner. This stage has limited influence of technology constraints. In a meeting or series of meetings the roadmap will be drawn by the product owner. This can be quite literally, through a graphical representation of the releases, or more formally in a written document outlining the dates, contents and objectives of the foreseen releases.

A product roadmap shows how the product will evolve over the next two to three releases. The product roadmap is a high-level representation of what features or themes are to be delivered in each release, the customer targeted, the architecture needed to support the features, and the business value the release is expected to meet.

3 Release Level

3.1 Release Vision/Scope Document

Release Vision/Scope Document defines release date, goal and planned features. It can also include any assumptions, dependencies, constraints, decisions made, concerns, risks, or other issues that may affect the release. Approved version is binding for project maneger. The Vision/Scope document is organized into sections:

- Business Opportunity: a description of the market situation and needs
- Solutions Concept: the approach the project team will take to meet the needs
- Scope: the boundary of the solution defined though the range of features and functions, what is out of scope, a release strategy, and the criteria by which the solution will be accepted
- Solution Design Strategies: the architectural and technical designs used to create the customer's solution
- Release tradeoff matrix: an agreement regarding the default priorities when making tradeoff decisions

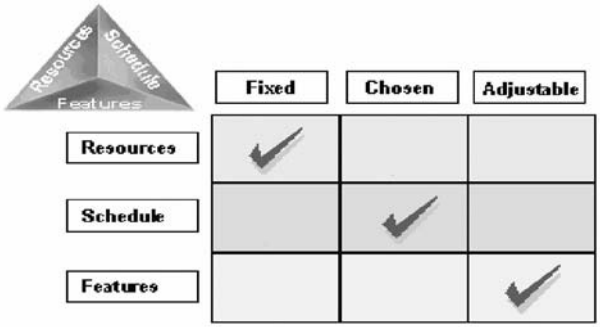


Figure 2

Release Vision/Scope document is written at the release planning level and is used for developing more detailed technical specifications and release plans. It provides clear direction for the project team; outlines explicit, up-front discussion of release goals, priorities, and constraints.

NOTE: The word "release" does not solely mean a product release to the end customer — it can also mean an internal release to fulfill integration milestones and continue to confirm that the product is "potentially releasable".

4 Iteration Level

4.1 Iteration Planning

During release planning the delivery of features was determined, taking dependencies between the features and between teams into consideration. Due to the lack of detail the estimates are rough and have an acceptable uncertainty.

For iterations within the release, a planning session takes place. During the session, detail is added to the features through decomposing into tasks. Adding detail increases accuracy of the estimates.

The agenda for the planning meeting bears great similarities with the one from the release planning meeting, with the prime distinction of the planning horizon: only a single iteration is observed in all activities. The core of the activities is carried out on a team-by-team basis:

- The individual teams determine their *actual* capacity, or the amount of work it can get done within the iteration.
- The individual teams decompose as many features as seem to fit in the next iteration into tasks — this can be done in preparation.
- Every task is estimated, with a typical task size of a half a day to two days.

5 Daily Level

5.1 Daily Plan

Our agile teams have taken on the habit of a daily meeting: the SCRUM stand-up meeting. In a 10-minute session, the team members update their status:

- what did I do yesterday,
- what will I do today,
- I am blocked by...

This daily meeting is not often seen as a planning session, but certainly is. The people look a day ahead, have learned from the earlier days in the iteration and tell each other what they plan on doing. Issues are raised, possibly addressed, and the success of delivering the desired features within the iteration can be determined after the meeting.

Like other planning activities, the daily plans need to be synchronized between teams. This takes place in a coordinating stand-up meeting ('Scrum of Scrums'). Representatives of each team report the status, plans and impediments to each other in an identical format. The three questions answered are the same as in the individual stand-up meetings

Extrémní programování

Martin Junek

E-mail: martin.junek@aimtec.cz

Abstrakt

Extrémní programování (XP) je promyšlený přístup k vývoji software. Od roku 1996, kdy byla tato metodika formálně popsána, ji ověřilo mnoho vývojových společností různých velikostí v různých odvětvích v celém světě. Vývojové oddělení společnosti Aimtec úspěšně využívá XP již několik let. Cílem této přednášky je nejen seznámit vás s principy XP. Dozvíte se také, jaké přínosy mělo zavedení metodiky pro efektivitu celého vývojového oddělení. Rozkryjeme také nejčastější problémy všech vývojových týmů, které se zdaleka netýkají pouze programátorů. Extrémní programování si klade za cíl nejen zefektivnit vývoj, ale má být také motivační pro celý tým. Jinými slovy, práce musí být především zábavná.

Autor příspěvěk nedodal.

Git aneb správa verzí trochu jinak

Jan Kasprzak

E-mail: kas@fi.muni.cz

Abstrakt

V oblasti systémů pro údržbu a správu verzí zdrojových kódů (SCM) došlo v posledních letech k bouřlivému vývoji, který je charakterizován přechodem od centralizovaných systémů typu CVS (a později Subversion) k plně distribu- ovaným systémům. Systém Git, představený v tomto příspěvku, nepatří mezi nejstarší mezi podobnými systémy, nicméně v mnohém se těmi staršími nechává inspirovat, a naopak v jiných případech si jde svou vlastní cestou. V příspěvku se zaměříme hlavně na vnitřní fungování Gitu, a z toho pak vyplynou možné způsoby použití.

Abstract

In the last few years, there has been an intense development in the area of source code management systems (SCM). This can be characterized mainly by the movement away from the centralized systems like CVS (and later also Subversion) to fully distributed systems. The Git system, introduced in this paper, is not the oldest amongst the similar systems, but it is inspired by the older systems in many ways, but it also uses its own way of doing things in several other cases. In the paper we will focus on the internals of Git, and this will lead us to possible use cases.

Klíčová slova: Version control sytems, distributed source code management, Git, Linux

1 Úvod

S pojmem *verzovací systém*[1] se setkal asi každý, kdo někdy pracoval poblíž vývoje softwaru. Pod tímto pojmem rozumíme software, do kterého lze ukládat verze zdrojových textů tak, aby bylo možno se případně v budoucnu mohli vrátit k některé starší verzi. Nejprve základní pojmy z oblasti verzovacích systémů:

repozitář (repository): místo, kam si verzovací systém ukládá informace o tom, jak vypadaly starší verze, a další svoje metadata.

checkout: zkopírování některého z uložených stavů z repozitáře do lokálního souborového systému.

pracovní kopie: data vyexportovaná z repozitáře akcí checkout.

commit: akce uložení změn v pracovní kopii do repozitáře. Někdy se však tímto pojmem označuje i verze dat tímto procesem vytvořená.

Ukládání verzí je základní funkce SCM systému. Dále verzovací systémy obvykle poskytují následující vlastnosti:

větvení: z jednoho stavu v čase může vycházet více paralelních větví vývoje (např. stabilní a vývojová větev). Je nutno říct, že podpora slučování větví (merge) je ve většině systémů velmi slabá.

komentáře: ke každé změně (*commit, changeset*) je možno připojit popis dané změny a případně odkaz na systém pro správu chyb.

paralelní přístup: většina systémů se nějakým způsobem snaží řešit přístup více vývojářů ke zdrojovým textům a případné zamykání.

háčky: skriptovatelné akce, vykonané v různých fázích práce s verzovacím systémem (například automatizované spuštění regresních testů při *commit*u do hlavní větve).

štítky: textové názvy verzí. Jednotlivé verze pak můžeme adresovat podle času vzniku, interního čísla verze ve verzovacím systému a se štítkem i podle štítku dané verze. Štítek může například odrážet fakt, že „tato verze byla zveřejněna jako 1.0.1“.

Verzovací systémy se mohou lišit v některých vlastnostech, jako například jestli podporují ukládání binárních nebo jen textových dat, jestli je možno ukládat metadata typu UNIXová přístupová práva, MIME typy souborů a podobně, jestli pracují nad jednotlivýcmi soubory (RCS, SCCS, CVS) nebo nad celým stromem adresářů (Subversion), detekcí porušení repozitáře, atd.

[1] alternativní pojmy: systém pro správu verzí, systém správy zdrojových textů, source code management system (SCM system), version control system (VCS)

2 Distribuované SCM systémy

2.1 Centralizované systémy

Až do poměrně nedávné doby měly všechny verzovací systémy *centralizovanou architekturu*. To znamená, že repozitář pro daný projekt byl na jednom serveru, ke kterému se jednotliví vývojáři připojovali, ať už s právem čtení (stahování verzí, výpis historie, atd.) nebo i s právem měnit data (právo *commit*). Vývojář si pak vždy stáhnul aktuální verzi na svůj stroj, provedl změny, a tyto pak promítnul do repozitáře.

2.2 Problémy centralizovaných systémů

Centralizovaný přístup s sebou nese několik problémů. Ten nejméně závažný je, že pro využití výhod verzovacího systému je třeba mít síťový přístup k repozitáři. Není tedy možno vzít si práci na chalupu a přitom kromě psaní programu dělat operace typu „zjisti kdo psal tenhle kus kódu“ nebo „oprav překlep v dokumentaci ve větvi, jejíž *checkout* jsem si neudělal předem“.

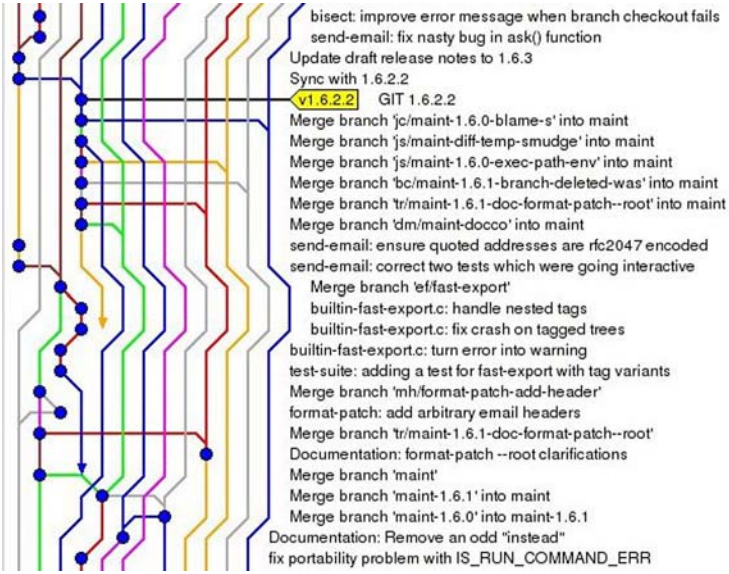
Tím se dostáváme k dalšímu problému: centralizované verzovací systémy nutí vývojáře, aby commit nedělali příliš často. Obvyklá praxe je, že vývojář udělá commit jen na konci pracovního dne. Jednotlivé commity ve verzovacím systému jsou pak čitelné jen jako „zkompilovatelné stavy projektu“, ale už ne jako logické k sobě patřící změny. V centralizovaném systému je poměrně těžké „odložit si“ právě rozpracovanou rozsáhlejší činnost a opravit triviální chybu nebo překlep ve formě samostatného commitu.

Asi největším problémem centralizovaných verzovacích systémů je překvapivě jejich centralizovanost. Centralizovanost znemožňuje například několika „okrajovým“ vývojářům bez práva zápisu do hlavního repozitáře dohodnout se a ad-hoc spolupracovat na nějaké části vývoje, včetně výměny svých částí kódu za pomoci verzovacího systému. Centralizované systémy tak přinášejí do projektů značnou míru politikaření a rozdělování světa na „core team“ s právem commit a „ty ostatní“, kteří mohou výhod verzovacího systému používat jen velmi omezeně.

2.3 Distribuované systémy

Distribuované SCM systémy představují kvalitativně novou úroveň práce s verzovacím systémem. Základní vlastností je neexistence jednoho centrálního repozitáře, do kterého mají přístup (někteří) vývojáři[2]. V distribuovaném verzovacím systému může existovat (a reálně také existuje) mnoho různých repozitářů. Na

[2]U většiny projektů samozřejmě „něco jako“ centrální repozitář existuje i nadále. Jeho status ale není dán funkčně (že by každý vývojář potřeboval mít k němu právo zápisu), ale deklarativně (je to *ten konkrétní* z mnoha možných a funkčně ekvivalentních).



Obrázek 1 Graf verzí v projektu Git

rozdíl od operace *checkout* u centralizovaných VCS je zde operace *clone*, která znamená vytvoření lokální kopie repozitáře, včetně veškeré historie.

Obvyklá metoda práce pak je vytvoření klonu nějakého repozitáře, práce uvnitř lokální kopie (s případným sdílením dat s jinými repozitáři) a zpřístupnění vlastních změn ve formě svého vlastního repozitáře, odkud si správce projektu (anebo kdokoli jiný) může dle svého uvážení změny vzít a použít.

Další rozdíl je pak ve způsobu nahlížení na data. Zatímco centralizované systémy považují jednotlivé verze za snímky dat v určitém čase[3], distribuovaný systém uvažuje v intencích posloupností *změn v datech* (angl. *changesets*). Celý vývoj pak je zobrazitelný nikoli ve formě posloupnosti, ale orientovaného acyklického grafu. Jednotlivé verze jsou pak charakterizovány nikoliv svým obsahem, ale seznamem změn (obvykle oproti společnému předkovi), které byly do dané verze zahrnuty. Příklad grafu změn v projektu s distribuovaným SCM je na obrázku 1. Komunikace mezi jednotlivými repozitáři může být jak nativní – prostředky konkrétního verzovacího systému – tak i obecná, například zasíláním souborů ve formátu programů *diff(1)* a *patch(1)*.

Významnou vlastností distribuovaných SCM systémů je, že typicky poskytují širokou škálu metod čtecího přístupu k repozitáři. Repozitářem pak může být

[3]V Subversion dokonce lineárně uspořádané do posloupnosti.

i libovolný statický adresář, zpřístupněný protokolem HTTP nebo FTP. Tímto je umožněno, že svoji vlastní práci může i s výhodami verzovacího systému zveřejnit opravdu každý, kdo má aspoň nějaký HTTP server. Není třeba ani právo zápisu do centrálního repozitáře, ani provozování vlastního serveru pro verzovací systém, ani vytvoření „oficiálního“ projektu například na SourceForge. Kdokoli takto může snadno zveřejnit svoje úpravy libovolného jinde dostupného projektu se všemi výhodami použití verzovacího systému.

3 Systém Git

3.1 Historie

Operační systém Linux byl poměrně dlouho vyvíjen bez použití verzovacího systému. Hlavní správce vývoje – Linus Torvalds – dlouho nechtěl žádný verzovací systém, protože pole něj by tehdejší systémy jej spíše brzdily než by přinášely zlepšení. Až v roce 2002 Larry McVoy, autor verzovacího systému pro potřeby Sun Microsystems TeamWare, přišel s projektem verzovacího systému BitKeeper, který byl jedním z prvních plně distribuovaných systémů. Linus Torvalds pak začal BitKeeper používat pro vývoj jádra Linuxu. BitKeeper ovšem nebyl Open Source, což vzbudilo nevoli některých dalších vývojářů.

Přibližně ve stejné době započal vývoj nástroje GNU Arch, který byl prvním Open Source distribuovaným verzovacím systémem. Arch ovšem měl několik praktických omezení (zejména výkon, z uživatelského hlediska pak poměrně složitý způsob práce a složité pojmenovávání verzí a větví), takže jeho nasazení v reálných projektech bylo spíše sporadické.

Dalším významným distribuovaným verzovacím systémem byl Monotone, jehož autoři přišli s myšlenkou, která do značné míry řešila problematiku celosvětově jednoznačného pojmenovávání sad změn (changesets) v distribuovaném prostředí. Nápad spočíval ve využití kryptograficky silné hashovací funkce pro označení verze souboru nebo i stromu souborů. Pouhým spočtením SHA1 součtu souboru můžeme celosvětově jednoznačně adresovat jeho obsah.

V dubnu 2005 Larry McVoy ukončil licenci BitKeeperu pro vývoj jádra Linuxu (důvodem bylo to, že autor Samby Andrew Tridgell zveřejnil výsledky svých pokusů o reverzní inženýrství síťového protokolu BitKeeperu) a Linux se ocitl zpět v bodě nula. Linus Torvalds se rozhodl na několik týdnů přerušit vývoj Linuxu a věnovat se vytvoření verzovacího systému. Torvalds na základě zkušeností s BitKeeperem a na základě informací o dalších verzovacích systémech navrhl systém Git [1]. I když z původní implementace toho v současném Gitu mnoho nezbylo, základní architektura, zajišťující extrémně rychlou odzevu, se používá dodnes.

4 Architektura Gitu

V následující kapitole si ukážeme principy, na jakých Git staví. Výklad je volně inspirován článkem [2]. Není to uživatelská příručka ani rychlokurz. Pro tyto typy dokumentace odkazujeme na [3], resp. [4].

4.1 Konfigurace

Pro práci s Gitem je vhodné, aby u změn byl uveden smysluplný autor. Toho dosáhneme příkazem `git-config(1)`, například:

```
$ git config --global user.name 'Jan "Yenya" Kasprzak'
$ git config --global user.email kas@fi.muni.cz
```

Git ukládá globální nastavení v souboru `.gitconfig` v domovském adresáři. Bez přepínače `--global` lze upravovat nastavení jen pro konkrétní repozitář.

4.2 Repozitář

Repozitář je v Gitu běžný adresář. Pokud vedle repozitáře mají být též vykopírovány zdrojové texty v nějaké verzi, říká se jim *pracovní kopie*. Repozitář pak má svoje data v podadresáři `.git` v kořeni pracovní kopie. Pokud pracovní kopii nepotřebujeme (například při zveřejnění repozitáře přes HTTP), jedná se o tzv. *holý repozitář* (*bare repository*), který je adresářem (obvykle) s příponou `.git`. Repozitář lze vytvořit buďto jako úplně nový (pro svůj vlastní projekt) nebo jako klon jiného repozitáře (pro úpravy existujícího projektu):

```
$ mkdir mujprojekt; cd mujprojekt; git init [--bare]
$ git clone [--bare] http://server/projekt.git; cd projekt
```

4.3 Blob, tree, commit

Git je v podstatě obsahově adresovatelná databáze souborů. Každý soubor je identifikován SHA1 hashem svého obsahu a je označen pojmem *blob*. Hodnotu SHA1 hashe není třeba uvádět celou, stačí libovolný prefix, který hodnotu jednoznačně identifikuje v rámci repozitáře:

```
$ echo ahoj > pozdrav.txt
$ git hash-object pozdrav.txt
36a0e7cf17ffe584221529d45113d821e70764a9
$ git add pozdrav.txt
$ git cat-file -p 36a0e7cf17
ahoj
```

Jak vypadal daný projekt v určité verzi je popsáno v objektu typu *tree*. Tento objekt obsahuje jména souborů které v daném čase do projektu patřily a SHA1 hashe jejich obsahů v dané verzi.

```
$ git write-tree
f1d1e2642787c59d61f01075e44b780e60ede900
$ git cat-file -p f1d1
100644 blob 36a0e7cf17ffe584221529d45113d821e70764a9   pozdrav.txt
```

S objekty typu tree obvykle uživatel nepřichází do styku přímo. Jednotlivé verze jsou identifikovány objekty typu *commit*. V nich se nachází odkaz na verze souborů (objekt typu tree), odkaz na rodičovský commit (nebo commity) a další metadata.

```
$ git commit -m 'Prvni verze'
Created initial commit f2ebc71: Prvni verze
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 pozdrav.txt
$ git cat-file -p f2eb
tree f1d1e2642787c59d61f01075e44b780e60ede900
author Jan "Yenya" Kasprzak <kas@fi.muni.cz> 1240789610 +0200
committer Jan "Yenya" Kasprzak <kas@fi.muni.cz> 1240789610 +0200

Prvni verze
```

Objekt typu commit v sobě již zahrnuje mimo jiné čas vytvoření, takže jeho SHA1 hash se na rozdíl od výše uvedených bude na vašem počítači lišit, i když budete výše uvedené příkazy zadávat doslova. První commit nemá žádného rodiče. V dalších je ale SHA1 hash rodiče nebo rodičů zmíněn:

```
$ echo nazdar >> pozdrav.txt
$ git add pozdrav.txt
$ git commit -m 'Pridan pozdrav "nazdar".'
Created commit b400b96: Pridan pozdrav "nazdar".
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git cat-file -p b400b96
tree f3424e549969ff3293ddb04970bde3128c836887
parent f2ebc71c3ca437c668c99731f6ae13efe1cb2f18
author Jan "Yenya" Kasprzak <kas@fi.muni.cz> 1240790025 +0200
committer Jan "Yenya" Kasprzak <kas@fi.muni.cz> 1240790025 +0200

Pridan pozdrav "nazdar".
```

Tímto SHA1 hash commitu jednoznačně identifikuje nejen commit samotný coby sadu změn proti rodiči, ale (protože commit zahrnuje i SHA1 hashe rodičů) celou historii zpětně až ke vzniku projektu. Toto je významná bezpečnostní vlastnost Gitu – není možno bez povšimnutí modifikovat zpětně historii. Navíc takto lze detekovat i porušení integrity repozitáře.

4.4 Větve, štítky

Jednou z nejsilnějších vlastností Gitu je práce s větvemi. Uživatel si může snadno vytvářet větve (i dočasné), kopírovat mezi nimi změny a spojovat větve mezi sebou. Větev není nic jiného než pojmenovaná *hlava* (*head*), tedy commit který nemá potomky. Při commitu dalších změn do větve se odkaz přepíše na nově vzniklý commit. *Štítek* (*tag*) pak je textový odkaz na jeden pevný commit. Větve jsou uchovávány uvnitř repozitáře v podadresáři `refs/heads`, štítky v `refs/tags`. Implicitní větev se jmenuje *master*.

```
$ git checkout -b japonsky
Switched to a new branch "japonsky"
$ git branch
  master
* japonsky
$ echo konnichiwa >> pozdrav.txt
$ git add pozdrav.txt
$ git commit -m 'japonsky pozdrav'
Created commit ce1bbf5: japonsky pozdrav
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git checkout master
Switched to branch "master"
$ cat pozdrav.txt
ahoj
nazdar
$ git merge japonsky
Updating b400b96..ce1bbf5
Fast forward
 pozdrav.txt |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
$ cat pozdrav.txt
ahoj
nazdar
konnichiwa
$ ls .git/refs/heads/
master  japonsky
$ git branch -d japonsky
```

5 Další vlastnosti Gitu

Na výše popsané architektuře Git staví poměrně silné uživatelské rozhraní. Není cílem v tomto článku popsat typické použití Gitu. V této kapitole zmíníme některé vybrané vlastnosti a principy, kterými se Git odlišuje od jiných podobných systémů:

5.1 Produktem je záplata

Jednou ze základních myšlenek Gitu je, že produktem práce programátora nemá být pouhý nový nebo opravený kus programu, ale že tímto produktem má být „záplata“, tedy rozumně vytvořený, zdokumentovaný a logicky izolovaný popis změny oproti předchozí verzi. S těmito záplatami je pak možno zacházet různě, například použít jen některé, měnit jejich pořadí a podobně. Tomuto cíli Git podřizuje i styl práce. Například v Gitu není historie cosi neměnného. Koneckonců jde stále o lokální soubory na programátorově disku. Čili dokud programátor nepublikuje svoji práci veřejně, měl by mít možnost upravovat svoje commity dle libosti.

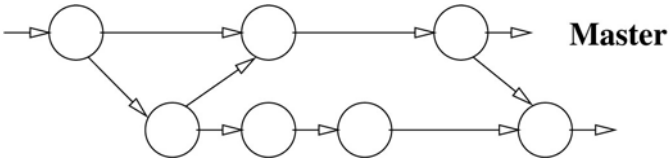
Častá je například situace, kdy programátor commituje novou funkci, ale zapomene ji popsat v manuálu. Git umožňuje pomoci příkazu `git-commit --amend` editovat poslední commit: přidávat do něho další změny, upravovat popis, a podobně. Nová verze commitu má pochopitelně nový SHA1 identifikátor, ale u dosud nepublikovaných commitů tohle nijak nevadí. Git umožňuje podobným způsobem editovat i starší commity.

Git má podporu pro export například celé větve jako sady záplat, poslání této sady e-mailem a na druhé straně aplikování několika mailů jako záplat do zadané větve.

5.2 Bisect

Máme-li jednotlivé úpravy jako logicky oddělené sady změn (a ne například sady změn typu „Moje práce za ten a ten den“), je i snadnější odhalování chyb na straně uživatele. Nástrojem `git-bisect(1)` je možno označit si která verze je funkční a která nefunkční, a Git pomocí půlení intervalu[4] umožní vytvářet mezilehlé verze, které pak uživatel otestuje a označí, jestli tato verze funguje nebo ne. S jistou dávkou štěstí pak může uživatel dospět k jedinému commitu, který daný problém způsobuje.

[4]Pojem „půlení intervalu“ je zde použit jen velmi volně. Připomínám že distribuovaný verzovací systém nemá změny jako striktně uspořádanou posloupnost, ale jako acyklický orientovaný graf.



Obrázek 2 Spojování větví – merge

5.3 Staging area

Pozorný čtenář znající i jiné verzovací systémy si jistě povšiml, že ve výše uvedených příkladech bylo použito příkazu `git-add` častěji, než by očekával. Je to kvůli vlastnosti Gitu, kterou se tento systém odlišuje od jiných verzovacích systémů: *staging area*, někdy též méně výstižně nazývaná *index*.

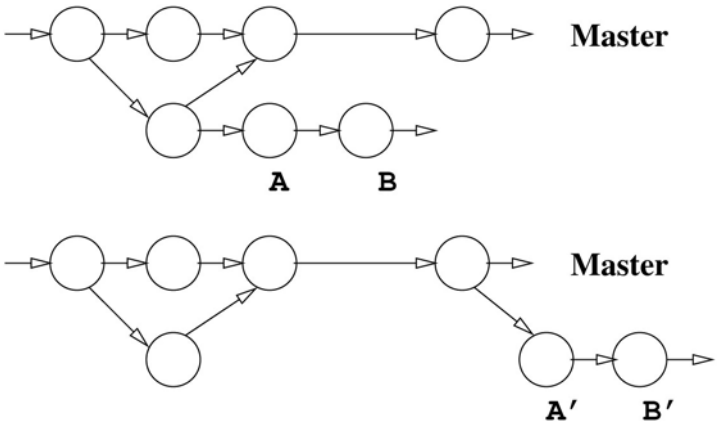
Příkaz `git-add(1)` zde má odlišný význam od příkazů `add` z jiných verzovacích systémů. Tam tato akce v podstatě znamená „začni sledovat tento soubor verzovacím systémem“. V Gitu tato akce znamená „do příštího commitu zahrň tento soubor s právě tímto obsahem“. Git kromě pracovních (vyexportovaných) dat a repozitáře již potvrzených (commit) dat má ještě třetí typ dat, a to staging area – data naplánovaná pro příští commit. Takto si programátor může commit konstruovat postupně po jednotlivých souborech. Do commitu tedy nejsou automaticky zahrnuty všechny modifikované soubory z pracovní oblasti, ale jen ty explicitně označené.

Pro uživatele přecházející z jiných verzovacích systémů má nicméně Git příkaz `git commit -a`, který dělá totéž co `commit` v jiných verzovacích systémech, tedy potvrzuje všechny změněné soubory.

5.4 Merge a rebase

Většina verzovacích systémů podporuje v nějaké formě slučování větví (merge). V Gitu je slučování větví poměrně jednoduché – hledají se všechny commity které jsou potomky společného rodiče větví a jsou předky těch hlav větví, které se právě slučují. Tyto se pak zahrnou do výsledných dat, jak je naznačeno na obrázku 2.

Mnohdy je však žádoucí svoje změny vystavit v podobě pro uživatele čitelnější, a to jako změny proti aktuální oficiální verzi (nazvěme ji třeba *master*). Git toto řeší pomocí akce *rebase*. Při rebase Git vezme změny z dané větve (které ještě nejsou obsaženy v master větvi) a vytvoří z nich nové commity se stejným obsahem, které ale naváže jako potomky hlavy větve master. Nové commity mají pochopitelně jiný SHA1 hash, protože se zejména liší jejich rodič. Situace je naznačena na obrázku 3.



Obrázek 3 Spojování větví – rebase

5.5 Stash

Chce-li vývojář mít nejen čitelný kód, ale i smysluplné a logicky oddělené commity, narazí občas na situaci, kdy by měl opravit triviální chybu v situaci, kdy má rozdělanou nějakou větší práci. Git pro tyto účely poskytuje nástroj *stash*:

```
$ vi pozdrav.txt # rozpracovana vetsi vec
$ git stash
Saved working directory and index state
"WIP on master: ce1bbf5... japonsky pozdrav"
HEAD is now at ce1bbf5 japonsky pozdrav
(To restore them type "git stash apply")
$ ... oprav trivialni chybu ...
$ git commit ...
$ git stash apply # vracim se k rozpracovane veci
```

Stash umožňuje mít i více takto rozdělaných a odložených prací. Nicméně u více rozpracovaných déletrvajících činností je spiš doporučeno dávat každou takovou činnost do samostatné větve.

6 Závěr

Mezi distribuovanými verzovacími systémy zaujímá Git významnou pozici. I když historicky je mu vyčítána slabší úroveň dokumentace, v posledních letech se tento problém definitivně odsunul do říše *urban legends*. Některé vlastnosti

jako je staging area mohou uživatelům jiných systému připadat neintuitivní, ale při častějším používání se stanou neocenitelnými pomocníky. Nedávné oznámení migrace několika velkých projektů (Perl, GNOME, Wine, X.org, ...) na Git činí z tohoto systému daleko nejrozšířenější distribuovaný systém správy verzí na světě. Budete-li začínat s novým projektem nebo uvažovat o změně verzovacího systému stávajícího projektu, zahrňte Git do svých úvah.

Reference

[1] *Domovská stránka projektu Git*
http://git-scm.com/

[2] *Wiegley, John: Git from the Bottom Up*
http://www.newartisans.com/blog_assets/git.from.bottom.up.pdf

[3] *Git User's Manual*
http://www.kernel.org/pub/software/scm/git/docs/user-manual.html

[4] *Git Tutorial*
http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html

Jasná a temná místa procesu testování v softwarovém projektu

Petr Žemla

E-mail: petr.zemla@upekka.net

Abstrakt

Příspěvek mapuje některá úskalí procesu testování, jak je navržen, řízen a vyhodnocován v praxi softwarového vývoje. Základ pro analýzu tvoří koncepty, na nichž jsou založeny metodiky rodiny „Unified Process“, zejména části týkající se testování, requirements, issue managementu a configuration managementu. Chápání a používání základních pojmů a postupů v oblasti testování softwaru není zdaleka tak stabilizované (relativně) jako v tradičních průmyslových disciplínách, dokonce ani jako v samotném vývoji softwaru. To vede v praxi k notorickému opakování problémů, které bývají ve své podstatě celkem jednoduché, ale mívají zásadní dopady na kvalitu produktu, dodržení termínů a jako konsekvence často i na vztahy uvnitř týmu. Přitom při včasném odhalení těchto typových problémů je lze často vyřešit nasazením poměrně jednoduchých opatření – dobrá zpráva obsažená v příspěvku se týká právě těchto osvědčených postupů a jednoduchých nástrojů.

V příspěvku uváděné historky čerpá autor zejména z vlastní praxe organizace a provádění testů. V rozsahu možném se dotkneme i výběru a nasazení nástrojů pro podporu testů.

Autor příspěvěk nedodal.