České vysoké učení technické v Praze

Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Jednotná autentizace v prostředí UNIX pomocí Public-key infrastructure

Bc. Daniel Slavík

Vedoucí práce: Ing. Jan Kubr

Studijní program: Elektrotechnika a informatika, kombinovaný magisterský

Obor: Informatika a výpočetní technika

Praha 2014

Poděkování

Poděkovat bych chtěl své rodině a zejména své sestře Petře a jejímu snoubenci Otovi, za neochvějnou podporu v průběhu celého studia a zvláště v posledních měsících. Dále bych chtěl podekovat mému zaměstnavateli, firmě AMI Praha a. s. za podporu a cenné rady kolegů. V neposlední řadě patří mé poděkování Ing. Janu Kubrovi, vedoucímu práce, za jeho vstřícný přístup a pomoc při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady v přiloženém seznamu. Nemám závažný důvod proti použití tohoto díla ve smyslu §60 Zákona č 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 12.5.2014 ……

Abstract

This masters thesis deals with the problem of secure access to the Linux/Unix servers via SSH. Problem of creating accounts and enable their security is common to every institution that owns a large number of these machines. Most of the applications run at this type of servers and the administrators access to the most of the servers using SSH, institutions must allow access to the all of the servers without compromising security. The beginning of the thesis deals with existing products and attemps to map out their advantages and disadvantages. The analytical part deals with the design of my own solution. In the implementation section I describe the creation of a "Proof Of Concept" application, the method of deployment and test the functionality of created aplication.

Abstrakt

Tato práce se zabývá problematikou zabezpečeného přístupu na Linux/Unix servery přes protokol SSH. Problém vytváření účtů a umožnění jejich zabezpečeného užívání je společný každé instituci, která vlastní větší počet těchto strojů. Protože na serverech tohoto typu běží většina aplikací, jejichž administrátoři přistupují na servery nejvíce pomocí protokolu SSH, je nutné umožnit všem přístup tak, aby nebylo narušeno zabezpečení serveru. V úvodu se práce zabývá již existujícími produkty a snaží se zmapovat jejich výhody a nevýhody. Analytická část se zabývá návrhem vlastního řešení. V implementační části je pak popsána tvorba „Proof Of Concept“ aplikace, způsob jejího vystavení a otestování funkčnosti.

Obsah

1 Úvod

Cílem této práce je navrhnout řešení pro správu účtů na systémech s operačním systémem Linux/Unix a umožnění jejich zabezpečeného používání. Problém správy účtů je společný každé instituci, která vlastní větší počet koncových systémů (serverů nebo aplikací). Oblast, řešící tuto problematiku, se nazývá Identity management a pro řešení problémů s ní spojených využívá aplikace zvané Identity manager. Obliba serverů a síťových prvků s operačním systémem Unix stále stoupá a díky tomu se stává jejich správa samostatným odvětvím tohoto oboru. Zvláště síťové prvky jsou z této problematiky často vymezovány, přestože je jejich zabezpečení pro jakoukoliv instituci klíčové a jejich vývoj v posledních letech umožňuje, aby k nim bylo přistupováno podobně, jako k jakémukoliv jinému Unixovému systému. Mnoha firmám se nově vyplatí spravovat Unix systémy jiným způsobem než ostatní servery a aplikace. Odlišná správa je způsobena jak zvyšujícím se počtem těchto systémů, tak i podporou rozličných technologií, které poskytují výhody nad standardním přístupem.

V úvodní studii jsem se pokusil popsat důvody, proč je potřeba centrální správa systémů a to jak správou uživatelů, tak správou oprávnění. Dále mapuji, jaké architektury se v dnešní době používají. Je zde analýza existujících řešením, architektur, výhod a nevýhod jejich nasazení.

Na základě výstupů z analýzy se v kapitole Návrh řešení pokouším navrhnout vlastní řešení dané problematiky. Navržená aplikace se bude zabývat správou uživatelů na Unixových systémech. Hlavním cílem aplikace nebude ale správa uživatelů, nýbrž umožnění zabezpečeného přístupu tak, aby byl pro uživatele přístup jednoduchý, pohodlný a pro obě strany komunikace bezpečný.

V poslední části práce je popsána tvorba „Proof Of Concept“ navrhnuté aplikace. Je zde popsán zvolený scénář, který je implementován i zvolený způsob implementace. Je zde také popsán způsob otestování implementovaného scénáře.

2 Úvodní studie

Jak již bylo v úvodu řečeno, obliba Unixových systémů ve světě serverů a síťových prvků roste. Díky jednoduchému vytváření virtuálních strojů, možnosti si systémy značně přizpůsobit a nízké ceně se servery Unixového typu pomalu dostávají do popředí statistických tabulek. Bohužel, přesto, že se správa Unixových systémů z pohledu Identity managementu značně podobá správě klasických aplikací, na mnoha místech a v mnoha institucích se k nim takto nepřistupuje. Jedním z hlavních důvodů je jejich počet. Počet Unixových serverů se v mnoha institucích blíží do řádů stovek až tisíců. Taková správa je pro klasický Identity manager velikou zátěží a způsobuje značné problémy. Dalším důvodem je časté opomíjení správy síťových prvků, které se děje jen velice zřídka. V neposlední řadě je klasickou cestou problematické spravovaní nestandardních (pro jiné aplikace) přístupů, jako je protokol SSH.

2.1 Definice pojmů

V následujícím textu se vyskytují, pojmy jejichž význam většina lidí obecně zná, ale pro zabránění nejasností zde budou jednoznačně pro zbytek textu definovány.

SSO (Single Sign On) - „Vlastnost systému umožňující prostřednictvím jediného přihlášení přiřazení přístup ke všem relevantním systémům.“

PKI (Public Key Infrastructure) – Je autentizační technologie používajíc šifrování pomocí veřejných a soukromých klíčů, jež byla poprvé definována v ITU-X.509.

2.2 Výhody centrální správy

Centrální správa uživatelů je důležitým prvkem zabezpečení každé firmy. Pokud je systémů více jak jednotky, je to i logickým krokem pro usnadnění práce administrátorů. Jednou ze značných výhod Unix systémů, je velká podpora neinteraktivních úloh. Bohužel to často vede k domněnce, že napojení na centrální správu aplikací a Identity managery není potřeba, protože si administrátoři mohou správu ulehčit a vlastně udělat sami. Největší problémy s tímto spojené jsem se pokusil popsat níže. Pro přehlednost jsou rozděleny podle úkonů, které se ve spojení s Identity managementem provádějí nejčastěji.

2.2.1 Vytváření a mazání účtů

V institucích se mohou vyskytovat až tisíce serverů. Každý server má v základním nastavení vlastní databázi uživatelů (/etc/passwd, /etc/shadow apod). Z toho plyne, že vytváření uživatele, který má přístup na stovky serverů, znamená vynaložení nemalého úsilí. V nejhorším případě se

musí administrátor (nebo administrátoři – při takovém množství je obvyklé fragmentovat zodpovědnost mezi více lidí, jak z důvodu množství, tak z důvodu časté heterogennosti systémů) přihlásit ke každému serveru jednotlivě a uživatele pomocí nativních příkazů uživatele vytvořit. Stejný problém nastává u mazání účtů.

Pro velkou pracnost a časovou náročnost si administrátoři často práci ulehčují pomocí ručně psaných skriptů. Tento způsob sice velice urychluje práci, ale má i různá úskalí:

- Skripty nemusejí doběhnout, protože na serverech nemusí být vše v pořádku (uživatel již existuje, skupina se stejným jménem nebo id již existuje apod.).
- Pokud dochází k úpravám důležitých serverových souborů, může dojít k značnému poškození počítače (například přepsání passwd prázdným souborem z důvodu střetu dvou paralelně pracujících operací).
- Snaha vytvořit uživatele s již existujícím UID.
- Nakopírování klíče do špatného adresáře po změně „sshd_config“ souboru.
- Mazání přihlášeného uživatele.
- Mazání uživatele s UID 0.

Pro tyto a mnohé další problémy se administrátoři snaží, aby se nastavení jednotlivých serverů lišilo co nejméně. Tento požadavek ale nemusí být slučitelný s potřebou aplikací, které na serverech běží.

2.2.2 Úprava oprávnění

Většina uživatelů, přistupujících na servery potřebuje vysoká oprávnění (číst důležité soubory systému, spouštět skripty, pracovat s daemony apod.). Oprávnění se ale mohou server od serveru znatelně lišit. V potaz přichází jak rozdělení serverů podle pracovního prostředí (uživatelé mají jiná oprávnění na vývojovém, testovacím prostředí, i na produkčním prostředí), tak druh aplikací, které na serverech běží (uživatel je administrátor jedné aplikace, ale u jiné jen kontroluje „log“ soubory). Oprávnění mohou být velice pestrá a to značné komplikuje vytváření jakýchkoliv administračních skriptů. V mnoha institucích se proto snaží oprávnění uživatelů co nejvíce zobecnit, což má za následek, že mnoho uživatelů má větší oprávnění, než ke své práci opravdu potřebuje.

Pokud neexistuje nějaký druh centrální správy, není jednoduché dohledat celková oprávnění uživatele. Jestliže je administrátor informován o změnách oprávnění uživatele (ať už emailem, nebo jiným druhem komunikace), je jen na jeho obezřetnosti, zda uživateli nastaví taková oprávnění,

které opravdu potřebuje. Není výjimkou, že se na některé účty zapomene. Tímto způsobem vznikají nekontrolované účty s vysokými oprávněními, u kterých není jednoduché dohledat, komu patří, nebo jestli jsou stále používány. Bez centrální správy je také téměř nemožné odhalit takzvané kolektory (uživatele, kteří na sebe nabalují značná oprávnění, aniž by je ke své práci potřebovali), kteří jsou

velkým nebezpečím pro každou instituci (a to nejen v případě personálních změn).

2.2.3 Udržování stavu účtů

Uživatelé s vysokým oprávněním mají často práva na vytváření, nebo úpravu již existujících účtů. Pro administrátora, který má na starosti stovky serverů, je takřka nemožné ověřit, že všechny účty na serveru jsou validní (žádný účet nepřibyl, všechny účty mají správnou platnost a nebyla změněna jejich oprávnění).

2.2.4 Přístup na server

Na servery se dá přistupovat mnoha způsoby, kde každý má své výhody a nevýhody. Nejčastějším a nejsnazším přístupem je heslo.

Největší nevýhodou tohoto typu přístupu je, že administrátor musí věřit, že uživatel heslo nevyzradí (vědomě, nebo nevědomě). Útočník by pak měl přístup k účtům uživatele, aniž by si uživatel, nebo administrátor něčeho mohli všimnout. Nejpoužívanější obranou proti tomuto druhu útoku je opakovaná změna hesla. Častá změna hesla je však z uživatelského hlediska velmi nepříjemná. Čím je politika hesel přísnější, tím je větší pravděpodobnost, že si jej uživatel bude někam zapisovat.

PKI řeší problém nevědomého vyzrazení. Uživatel jen těžko nevědomě vyzradí 2MB velký soubor. Problémem však zůstává možnost jeho odcizení. Administrátor musí věřit uživateli, že k jeho pracovní stanici nemá přístup nikdo jiný, že si jí uživatel zamyká při jím, třeba i krátkém, opuštění a že se klíč nachází na bezpečném místě na disku (a ne např. na flash disku kolujícím po celé firmě).

2.2.5 Shrnutí

Administrace většího počtu serverů je bez centrální správy identit velice problematická. Administrátoři mohou rychle ztratit přehled nad účty na serverech. Vytváření, úpravy a mazání účtů se značně prodlužují a samotná správa se komplikuje. Tím vším se administrace nezanedbatelně prodražuje. Bez centrální správy se také zvyšuje riziko vzniku nekontrolovaných účtů, které jsou vstupní branou do systémů instituce.

2.3 Architektury centrální správy

Existuje mnoho systémů pro centrální správu jak Unixových, tak i jiných systémů. Konkrétní případy jsou popsány v kapitole Existující produkty. Pro jednodušší popis těchto aplikací se pokusím vysvětlit základní principy architektur, které jsou v aplikacích implementovány.

2.3.1 Man in the middle

Toto je nejčastěji používaná architektura v komerčních produktech. Jedná se o případy, kdy se uživatel přihlašuje k proxy serveru a ten se přihlašuje k cílovému stroji za něj. Díky tomuto způsobu je proxy server schopen detailně monitorovat činnost uživatele. Velkou výhodou je, že uživatel nemusí znát způsob přístupu ke koncovému serveru (nezná heslo). Řešení je také neinvazivní k připojovaným serverům. Připojované servery o proxy serveru nemusejí nic vědět, protože se k nim proxy server přihlašuje standardní cestou. Dále je velkou úsporou počet účtů na serveru. Na serveru se mohou vyskytovat jen typové účty, které jsou používané jako sdílené. Detailní rozdělení účtů se nachází na proxy serveru.

Nevýhodou tohoto řešení je, že proxy server musí mít uloženy přístupové údaje k připojeným serverům v rozluštitelné formě. Veškerá komunikace také musí probíhat přes proxy server, který je tak velice zatížen. Při pádu proxy serveru se znemožní přístup na jakékoliv připojené servery.

2.3.2 Centrální database

Centrální databáze je typem architektury, kdy se všechny servery připojí k jedné databázi (LDAP, Active Directory, atd.). Při každém přístupu se server dotáže databáze, jestli má uživatel na daný přístup právo. Výhodou tohoto řešení je snadná správa uživatelů. Uživatelé jsou evidováni na jednom místě a jakákoliv změna v centrálním prvku se okamžitě projeví na všech připojených serverech. Samotná evidence serverů (passwd, groups, shadow, gshadow) je nahrazena databází. Tím je zajištěn jednotný přístup k serverům z administrativního hlediska (servery mají stejné skupiny se stejným GID a uživatelé mají stejné UID).

Nevýhodou tohoto řešení je, že při výpadku centrální databáze je znemožněn jakýkoliv přístup na připojené servery. Přístup je znemožněn i při větším zatížení sítě nebo větší ztrátovosti paketů. Na všech připojovaných serverech je nutné udělat změny (nastavení pam modulů, konfigurace serverů apod.), které nemusejí být jednoduché.

2.3.3 Centrální výdej tokenů

Tato architektura je zjednodušením obou předcházejících architektur. Uživatel musí komunikovat s centrálním serverem, aby mu server mohl vydat časově omezený token (může

jím být i heslo s časovou platností). Centrální server se pomocí komunikace s připojenými systémy snaží zajistit, aby se uživatel mohl pomocí tokenu na daný systém po určený čas přihlásit. Samotnou autentizaci si však připojené systémy řeší na své straně.

Hlavní výhodou tohoto řešení je značné omezení síťové komunikace a menší závislost připojených systémů na centrálním serveru. Pokud centrální server přestane pracovat, dotkne se výpadek jen uživatelů, kteří si chtějí o token teprve zažádat. Ostatní uživatelé mohou pracovat bez omezení. Nevýhodou tohoto řešení je nemožnost kontroly uživatele tak, jak je tomu u architektury Man in the middle.

2.4 Architektury přístupu

Architektury se také liší také způsobem, jakým umožňují uživatelům, aby se na servery přihlásili. Nejčastější způsoby jsou poptány níže

2.4.1 Heslo

Přihlašování pomocí hesla široce rozšířený a často podporovaný způsobů přihlašování. Pro zajištění nevyzrazení hesla jsou v produktech použity takzvané Password Managers. Tyto programy se starají o vydávání a změnu hesla. Heslo je uživateli vydáno jen na vyžádání na zadanou dobu. Po uplynutí této doby je heslo zase změněno. Tento postup se nejčastěji využívá v architektuře „man in the middle“, kde je heslo periodicky měněno i přesto, že jej uživatel vůbec nezná. Problém používání hesel je, že je uživatelé mohou snadno vyzradit. U neprivilegovaných účtů je běžné, že si hesla spravují sami uživatelé. Zde se ale udržení zabezpečeného stavu velice komplikuje. Požadavky na komplexnost hesla nebo jeho vynucená změna je uživateli vnímáno velice negativně. Z toho důvodu je nelze dělat tak často, jak by bezpečnost vyžadovala.

2.4.2 Token

Mnoho aplikací používá k přístupu takzvané tokeny. Jedná se o druh kódu s omezenou časovou platností, jehož validitu kontroluje buď připojovaný server, nebo samotná aplikace. Příkladem těchto tokenů může být bezpečnostní řešení pomocí SMS. Uživateli je při pokusu o přihlášení zaslán jednorázový kód, jehož správnost je pak zpětně kontrolována aplikací. Pokud je kód správný, a byl zadán v dostatečném čase (často je udáván čas jedné minuty), je uživatel přihlášen na server. Výhodou tohoto řešení je, že útočník musí buď ukradený token využít v časovém limitu, nebo odcizit celé zařízení pro předávání tokenů. Jako token může sloužit jednoduché heslo nebo jakýkoliv druh kódu.

2.4.3 Asymetrické šifrování

Základní myšlenkou asymetrického šifrování je rozdělit pomyslné heslo na dvě části. Pokud se zpráva zašifruje pomocí jedné části klíče, dá se rozšifrovat jen za pomoci druhé části klíče. Tím je serveru umožněno, že i přesto, že nezná první část hesla uživatele, je schopen ověřit jeho totožnost. Samotná historie a vývoj je popsána v kapitole Historie asymetrického šifrování.

2.5 Komerční produkty

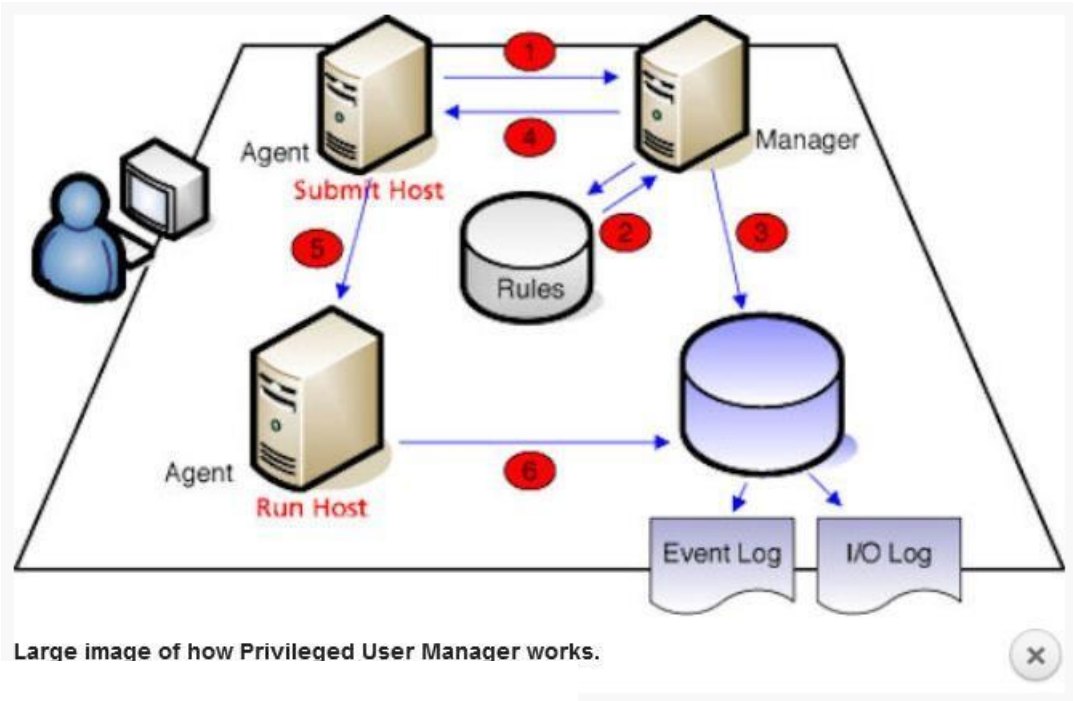
2.5.1 liebsoft – Privileged Account Password Management Solutions

Jedná se o řešení realizující správu hesel na připojených endpointech (koncových systémech). Hesla privilegovaných účtů jsou periodicky měněna tak, aby bylo znemožněno jejich vyzrazení. Pokud je účet potřeba použít, je heslo operátorovi vydáno a po uplynutí zadané doby od vydání klíče je heslo znovu změněno. Toto řešení je velice jednoduché na nasazení. Nejde o správu účtů v pravém slova smyslu. Produkt účty nevytváří, nemaže ani nemění jejich status. Dochází pouze k hlídání a periodickému měnění hesla.

2.5.2 NetIQ - privileged-user-manager

Správa privilegovaných účtů pomocí „man in the middle“. Každý příkaz, který chce uživatel na službě provést, je zachycen na proxy serveru a poslány na „Manager“ server. Zde je příkaz analyzován (porovnán s databází zakázaných příkazů a porovnán s nadefinovanými pravidly). „Manager“ server také zaloguje každý příchozí příkaz i s výsledkem analýzy. Pokud je analýza příkazu v pořádku a uživatel má na jeho spuštění oprávnění (oprávnění jsou i časová), je příkaz

vykonán na cílovém serveru. Výsledek příkazu je zalogován a vrácen uživateli.



Architektura produktu [24]

Pravidla uložená v databázi jsou vytvářena administrátory aplikace. Jejich počet ani komplexnost není omezena. „Manager“ server je také schopen auditu práce uživatele používající protokol FTP.

2.5.3 CA ControlMinder Shared Account Management

Aplikace vytváří sdílené účty. Ty jsou uživateli zpřístupněny přes webový portál. Portál se, po přihlášení uživatele do aplikace, přihlásí na koncový server za uživatele a zprostředkuje mu přístup. Jedná se o řešení man in the middle za použití hesel. Hlavní výhodou tohoto řešení je logování uživatelských akcí, nad kterými jsou prováděny sofistikované reporty pro rozpoznání hrozby. Reporty jsou zaměřeny na rozpoznání nestandardního chování vlastních pracovníků firmy, kde je produkt nasazen.

Aplikace umožňuje i přímý přístup k serveru. Server ale musí být připojen k tzv. „AuthMinderu“. Jedná se o podčást aplikace, která detekuje připojení k serveru. Aplikace pošle uživateli jednorázové heslo a server jej po zadání pošle aplikaci zpět (stejné řešení jako v případě RSA). Tento druh autentizace vyžaduje nainstalování klienta na server.

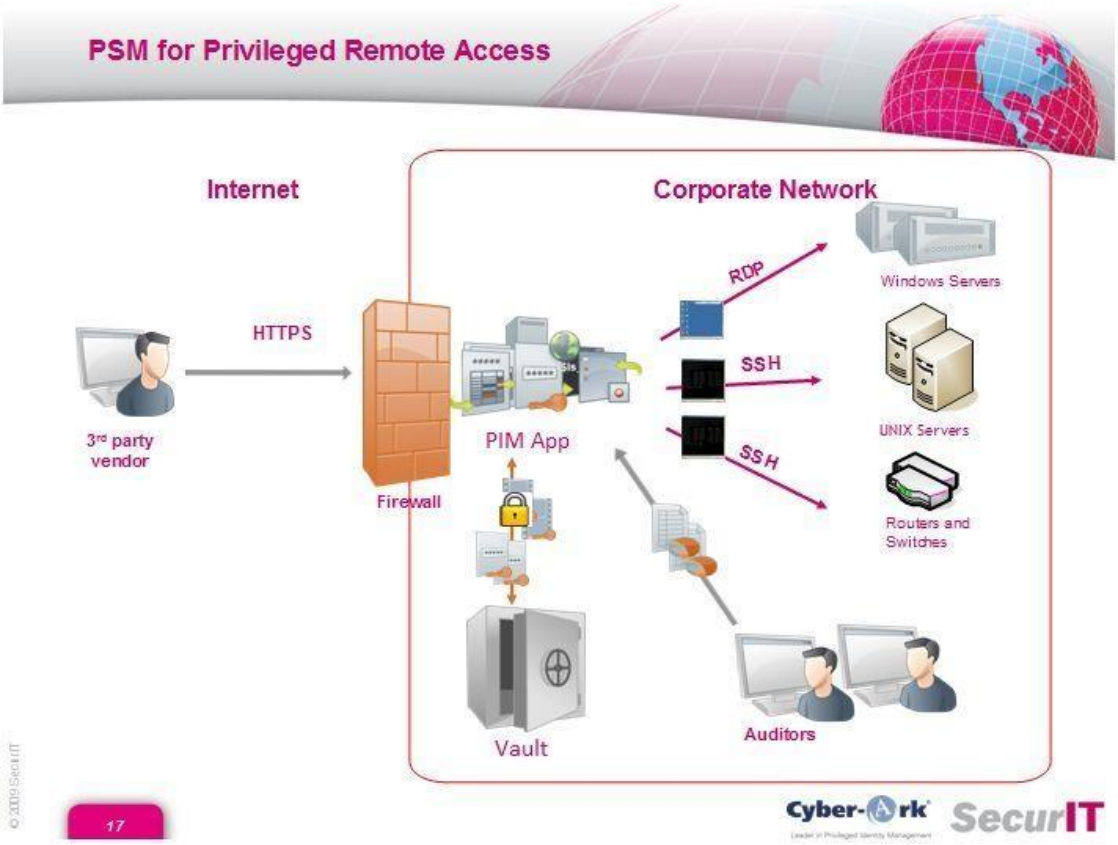
2.5.4 Beyondtrust - PowerBroker UNIX & Linux

Další řešení pomocí man in the middle. Proxy servery nekontrolují celou session. Pokud chce administrátor vykonávat nebezpečné operace, musí použít před příkazem „pbrun“ nebo zapnout

monitorovaný shell „pbrun bash“. Po této události má uživatel práva „root“, ale veškerá jeho aktivita je logována na server.

2.5.5 CyberArk - On-Demand Privileges Manager™ for Unix/Linux

Řešení za pomoci architektury man in the middle. Všechna hesla jsou uložena v patentovaném trezoru. Služba loguje veškeré aktivity uživatele a ukládá je do vlastní databáze. Logy jsou zpřístupněny auditorům přes speciální oprávnění ve webovém portále. Komunikace je šifrována pomocí certifikátů.



Architektura produktu [25]

2.5.6 Oracle Privileged Account Manager

Řešení využívá architekturu „man in the middle“. Jednotlivé účty jsou identifikovány za pomocí konektorů, které zprostředkovávají komunikaci se servery. Hesla k těmto účtům jsou uloženy v DB zašifrované pomocí Transparent Data Encryptoru. Konektory jsou vytvořeny pomocí ICF frameworku, který je hojně používaný v aplikacích pro správu identit takzvaných „Identity Managerech“. Díky tomu je poměrně jednoduché vytvářet nové konektory a připojovat jak

servery s nestandardním OS, tak i různé aplikace.

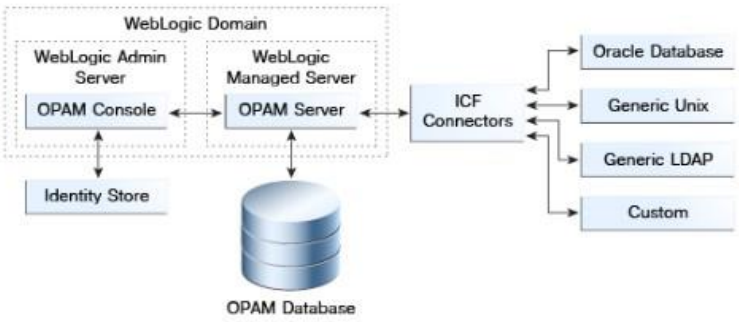


Figure 1: OPAM Architecture

Architektura produktu [26]

2.5.7 Centrify DirectManage

Řešení za pomoci využití již existující firemní infrastruktury (Active Directory dále jen AD). Veškeré Linux/Unix stroje jsou nastaveny tak, aby se dotazovali centrálního bodu při autentizaci. Na každý stroj je nutné nahrát agenta, který připojuje stroj k AD a překládá komunikaci (AD si myslí, že se jedná o stroj s operačním systémem Windows). Uživatelé se přihlašují pomocí hesla. Agenti na strojích zajišťují ukládání přístupových údajů pro případ nedostupnosti AD.

2.5.8 RSA Security

RSA security využívá architektury „centrální databáze“. Na rozdíl od jiných známých produktů ale nepoužívá k autentizaci heslo, nýbrž vlastní token. Autentizace má dva druhy:

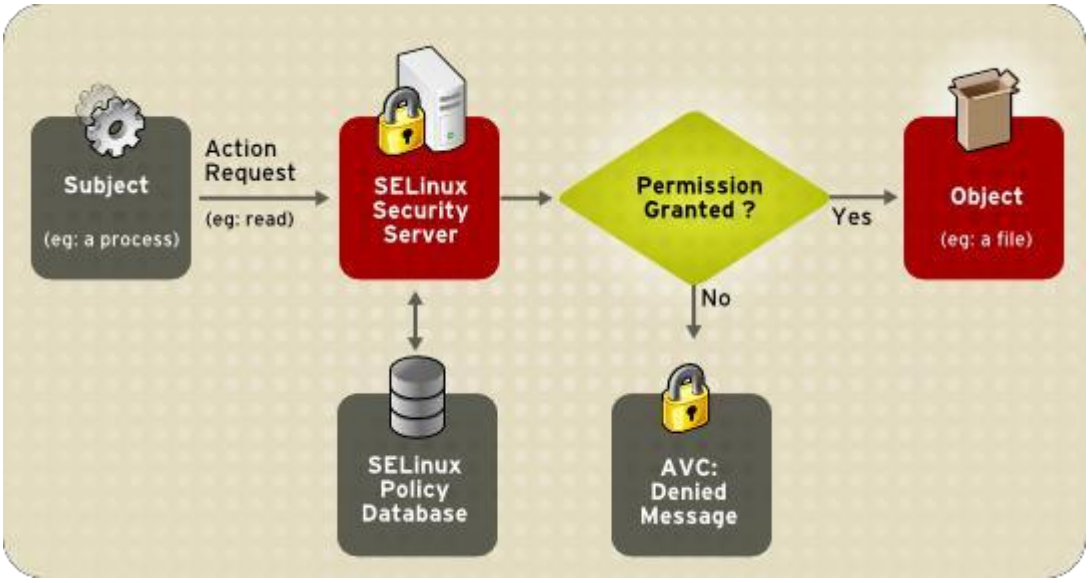
1. uživatel má vlastní token (přístroj generující každou minutu nové číslo) – Uživatel místo hesla při přístupu na server zadá vlastní pin (osmiznakové heslo) + šest číslic, které mu ukazuje token. V tokenu běží algoritmus, který každou minutu mění číslo na display. Stejný algoritmus běží na RSA serveru. Pokud uživatel zadá správný pin i správný kód z tokenu, je vpuštěn na server. RSA server je schopen podle správně zadaného kódu z tokenu poznat, o který konkrétní token se jedná a tím zalogovat i konkrétního uživatele.
2. uživatel vlastní token nemá – Autentizace má dva kroky:
    a. Uživatel zadá uživatelské jméno
    b. Na mobil je uživateli zaslán kód, který má platnost jednu minute
    c. Uživatel zadá svůj pin + zaslaný kód

Pokud je kód + pin správný a mezi zasláním kódu a jeho zadáním do serveru neuběhla více než minuta, je uživatel vpuštěn na server.

2.6 Volně dostupné produkty

2.6.1 SELinux

Rozšíření linuxového jádra. Soubor práv a operací. Vlastních rutin v kritických procesech jádra. Umožňuje nastavovat oprávnění jak pro uživatele, tak pro jednotlivé procesy nebo sokety. Chrání před prováděním potenciálně nebezpečných operací i po jejich vlastním spuštění (dokáže zabránit již běžícímu procesu v nebezpečné operaci). Umožňuje nastavit jemnější práva na soubory a procesy (jedná se o druh implementace MAC - mandatory access controls) a umožňuje definovat politiky, které jsou pak pomocí rutin vynucovány.



Architektura produktu [27]

Selinux opatřuje soubory, procesy, síťové zdroje apod. novými atributy (jako je například type). Pomocí těchto atributů může být nastaveno procesům nebo uživatelům přístup do specifikovaných souborů aniž by muselo dojít ke změně práv na souboru jako je skupina nebo vlastník nebo tam, kam by proces jinak přístup mít neměl (např. domovský adresář uživatele).

Selinux se využívá hlavně v obraně proti takzvanému „útoku nultého dne“. Tento útok spočívá v tom, že útočník nalezne chybu v již nainstalovaném programu a server je tak do vydání opravy nechráněn. Administrátor má v tomto případě jen velmi malou možnost útoku zabránit a ještě menší útoku předejít.

2.6.2 AppArmor

Rozšíření pro linuxové jádro. Na kritických místech jádra spouští své rutiny, pomocí kterých může testovat, co se systém snaží provést. Pomocí tohoto přístupu je možné zamezit uživatelům I procesům, aby poškodili daný server. Nevýhodou je zpomalení celého systému.

2.6.3 GRSecurity

Grsecurity je řada rozšíření pro linuxové jádro. Vylepšení implementují různé druhy ochran samotného systému.

Součástí grsecurity je například role-based access control (RBAC) pomocí kterého lze velice přesně nastavovat práva jednotlivých procesů nebo uživatelů. Pomocí RBAC lze vytvářet vlastní druhy oprávnění, takzvané role a ty přiřazovat uživatelům. Jednotlivé role se skládají z oprávnění, sepsaných v „Access control list“.

Další součástí grsecurity je PaX. Jedná se o balíky vylepšení pro jádro linuxu, které řeší bezpečnost paměti. Omezuje programy jen na svoji přidělenou paměť. Zabraňuje náhodnému čtení paměti při hledání pomocí odhadnutí adresy apod.

2.6.4 LDAP

LDAP není samostatné řešení pro zabezpečení přístupu. Jedná se o centrální prvek, na který lze migrovat passwd/shadow jednotlivých serverů. Využívá se pro vlastní implementaci architektury centrální databáze. K připojení stačí editace několika konfiguračních souborů:

1. /etc/ldap.conf
2. /etc/nsswitch.conf
3. /etc/pam.d/system-auth

Při správné konfiguraci se systém při každém pokusu o přihlášení dotazuje LDAPu a vyhledává v něm jako by se jednalo o soubory passwd nebo shadow.

2.6.5 Kerberos

Kerberos je protokol určený pro ověření identity jak uživatele, tak serveru, ke kterému se uživatel přihlašuje. Pracuje na základě předávání klíče asymetrického šifrování za pomoci časově omezených tiketů. K tomu, aby mohl bezpečně ověřit identity obou komunikujících stran, využívá důvěryhodného serveru (často nazývaného KDC – „key distribution center“). K ověřování uživatelů jsou využívána hesla. Server KDC generuje časově omezené tikety, které dokáže rozšifrovat pomocí svého hesla jen jedna komunikující strana a tím distribuuje šifrovací klíče. Protokol tak umožňuje zabezpečenou komunikaci na nezabezpečené síti. Protože je distribuce

závislá na serveru KDC jedná se o architekturu Centrální databáze. Protokol Kerberos je podporován mnoha systémy napříč operačními systémy. Pokud existuje správa hesel na koncové stanici uživatele, dá se poměrně lehce naimplementovat SSO pro všechny takto spravované systémy.

2.6.5.1 Nevýhody Kerbera

Přesto, že je Kerberos velice dobrou volbou pro autentizaci na koncové systémy, přináší jeho implementace několik nevýhod.

1. Nevýhody architektury Centrální databáze.
2. Nutnost časově synchronizovat všechny připojené servery (čas by se neměl lišit o více jak minut).
3. Kerberos není dobře snášen s PAM moduly.
4. Kompromitováním dat serveru KDC získává útočník možnost se vydávat za jakéhokoliv uživatele.
5. Připojení unixových systémů pod správu Kerbera je časově náročné (migrace souborů /etc/passwd a /etc/shadow)[28]

2.7 Problémy stávajících řešení

2.7.1 Používání hesel

Aby se firmám vyplatil vývoj aplikace, snaží se do ní připojit co nejvíce existujících systémů. Z tohoto důvodu se často musí omezit na autentizaci pomocí hesel. Autentizaci pomocí hesel v dnešní době nelze považovat za bezpečný způsob. U hesel nelze efektivně vynucovat častou změnu, aniž by uživatelé přestali aplikace používat. Hrozí jejich duplicita v rámci jiných uživatelových přístupů (hlavně těch soukromých), a díky tomu může dojít k vyzrazení bez chyby na straně uživatele, nebo infrastruktury instituce.

2.7.2 Hardwarové nároky

Firmy jako CyberArk nebo CA si problém používání hesel zajisté uvědomují a to je také jeden z důvodů, proč své aplikace vybavují propracovanými auditními systémy. Online kontrola umožňuje sice zabezpečení celého systému jak proti útoku vlastního zaměstnance, tak proti útoku pomocí zneužití identity, má ale značné vedlejší účinky. Z nutnosti zachytávat a analyzovat veškerou komunikaci uživatele se serverem je komunikace vedena přes proxy server. Z tohoto důvodu se firmy tak často uchylují k architektuře „man in the middle“. Proto má každá taková aplikace obrovské hardwarové nároky.

2.7.3 Kritický bod

Aplikace se většinou přihlašují za uživatele, nebo se každý server aplikací doptává, zda má uživatel právo na přístup. To znamená, že např. v případě použití aplikace „man in the middle“ musí mít server naráz otevřeny stovky až tisíce SSH spojení. V případě centralizované DB musí být umožněno, aby k aplikaci přistupovaly stovky požadavků najednou. Tímto způsobem se aplikace stává kritickým bodem celé infrastruktury. Při pádu aplikace je znemožněna práce všem uživatelům systémů, které jsou k ní připojeny. Tento problém se může týkat i nároků na síťovou infrastrukturu.

2.7.4 Cena

Samotná cena licencí pro aplikace, založené na architektuře „man in the middle“ se pohybuje v řádu milionů. Po nainstalování je také nutné platit roční licence.

Většina těchto řešení zabezpečuje servery, vytváří uživatele, vytváří velice sofistikované logování. Všechny tyto operace vyžadují značnou investici do infrastruktury. Aplikace běží na výkonných serverech. Je potřeba databázový server a investice do síťové infrastruktury.

2.8 Shrnutí

Existuje mnoho produktů, které se starají o správu účtů na různorodých systémech, pomocí různorodých technik. Mají ale problémy (popsané výše) které umožňují vstoupit na trh s novou úzce zaměřenou aplikací. Protože se produkty snaží spravovat co největší počet aplikací a operačních systémů, musejí se omezit na technologie, které jsou všem společné, nebo přijít s vlastní implementací nějakého druhu tokenu. V prvním případě (nejčastěji používaném) se musí produkty spokojit s používáním hesel. V druhém případě jsou nutné úpravy na straně všech připojovaných serverů (tokeny nejsou v základním stavu serverů podporovány).

Asymetrické šifrování se stává nedílnou součástí našeho světa. Jak analýza ukazuje, v oblasti Identity managementu se asymetrické šifrování zatím moc neujalo, i když se vyskytuje na webových stránkách, je součástí podepisování emailů nebo elektrických dokumentů. Asi hlavním důvodem je jeho malá podpora v rámci používaných protokolů. Tato nevýhoda ale v unixovém světě odpadá. Zde je podpora asymetrického šifrování značná. A pokud se má navrhovaná aplikace zabývat hlavně správou výhradně unixových systému, je asymetrické šifrování asi nejlepší volbou.

Na trhu je nyní velice málo aplikací starajících se jak o správu identit uživatelů, tak o nastavování jejich oprávnění a zajišťování zabezpečeného přístupu (nepoužívající heslo).

2.8.1 Shrnutí komerčních produktů

Komerční produkty se velice často zaměřují čistě na privilegované účty. Často se také předpokládá, že instituce, která je provozuje, již má implementované řešení pro správu identity a proto mnoho produktů ani na připojených systémech účty nevytváří. Produkty se snaží buď zabránit uživatelům před spuštěním nebezpečných příkazů, o vyhodnocení, jestli se uživatel nepokusil poškodit instituci (vyhodnocováním logů), nebo o zabránění přístupu nepovolaným uživatelům na koncový systém. Výsledek analýzy jsem se pokusil shrnout v následující tabulce

Název	Architektura	Správa identit	Správa oprávnění	Správa přístupu	Výhody	Nevýhody
Liebsoft – Privileged Account Password Management Solutions	Centrální výdej tokenů	Ne	Ne	Ano	Snadná instalace, jednoduché připojení systémů, cena.	Používání hesel
NetIQ - privileged-user-manager	Man in the middle	Ne	Ano	Ano	Detailní audit práce uživatele před odesláním příkazu na server.	Cena, jen pro privilegované uživatele.
Beyondtrust - PowerBroker UNIX & Linux	Man in the middle	Ne	Ano	Ano	Jednoduchá instalace, validace příkazů před odesláním.	Cena. Jedná se jen o validaci příkazů.
CyberArk - On-Demand	Man in the middle	Ano	Ano	Ano	Aplikace se příhlašuje	Cena, hardware

Privileges Manager™ for Unix/Linux					jiným heslem k server, než uživatel k aplikaci, patentovan é ukládání hesel, kompletní logování session uživatele, podpora mnoha protokolů.	nároky.
Oracle Privileged Account Manager	Centrální databáze	Ne	Ne	Ano	Využití konektorů Oracle, jednoduché napojení na IdM, správa mnoha různých systémů.	Potřebuje IdM, cena, používání hesel, nemožnost paralelního používání účtů.
RSA Security	Centrální databáze	Ne	Ne	Ano	Využití tokenů.	Cena, složité napojování systémů.

Analýza shrnutí placené 1

2.8.2 Shrnutí volně dostupných produktů

U těchto produktů se většinou nejedná o snahu vyřešit problematiky správy identit. Uvádím je zde, protože se dají v návrhu využít a mnoho firem je používá ve vlastních řešeních.

Jméno	**Správa identit**	**Správa oprávnění**	**Správa přístupu**	**Výhody**	**Nevýhody**

SELinux	Ne	Ano	Ne	Možnost omezit supersprávce, detailní nastavování oprávnění.	Zatěžování koncových systémů.
AppArmor	Ne	Ano	Ne	Možnost omezit supersprávce, detailní nastavování oprávnění	Zatěžování koncových systémů
GRSecurity	Ne	Ano	Ne	Možnost omezit supersprávce, detailní nastavování oprávnění	Zatěžování koncových systémů
Centrální LDAP	Ano	Ano	Ano	Jednoduchá implementace, kompletní řešení problematiky, cena	Složité připojování systémů, vysoká závislost na centrálním server, používání hesel, potřeba správy identit
Kerberos	Ne	Ne	Ano	Používání tokenů, cena, podpora v protokolech, SSO	Centrání server, kompromitace dat, připojování

	koncových systémů.

Analýza shrnutí neplacené 1

3 Vlastní návrh

3.1 Definice pojmů návrhu

V následujícím textu bude popsána navržená architektura. Pojmem daemon bude myšlena vždy serverová část architektury. Pojmem klient (nebo tlustý klient) bude myšlena aplikace instalovaná na klientské počítače. Celá architektura a tedy obě zmíněné části se v textu budou nazývat aplikace.

3.2 Režie aplikace

Výhody a nevýhody architektur pro centrální správu jsou popsány výše. Zde navržená aplikace bude podporovat dvě ze tří architektur. Jednou bude Centrální výdej tokenů a druhou bude Centrální databáze. Hlavní výhodou architektury Man In The Middle je možnost detailního zaznamenávání operací provedených na koncovém systému. Protože navrhovaná aplikace nebude takové logování provádět, není pro použití této architektury důvod. Důvodem pro Centrální výdej tokenů je odstranění prvku, který při své nedostupnosti znemožní přihlášení jakémukoliv uživateli. Tato vlastnost může být klíčovou pro nasazení aplikace u zákazníka. Centrální databáze bude podporována z důvodu režie. Za režii bude v následujícím textu uvažováno jen zatížení sítě při komunikaci mezi koncovým systémem a daemonem aplikace a to jen v závislosti na operacích potřebných k přihlášení uživatele. Ostatní operace jsou pro architektury shodné a proto je není nutné do režie, která se zabývá porovnáváním architektur, započítat.

Daemon aplikace se bude na koncové systémy připojovat jednou denně pro zkontrolování stavu konfiguračních souborů a to nezávisle na zvolené architektuře.

3.2.1 Centrální výdej tokenů

V této architektuře bude daemon rozesílat aktuální klíče v nastavenou hodinu. Daemon bude s klíči provádět dvě operace:

1. Kontrolovat, že se klíč nezměnil.
2. Rozesílání aktuálního klíč.

Pokud by první operace zjistila změnu klíče, znamenalo by to, že klíč byl změněn třetí osobou. To by znamenalo bezpečnostní incident. Není důvod předpokládat, že by se takové incidenty děly často a proto nebude operace spojená s opravou klíče připočítána do režie architektury. Režie architektury pro umožnění přihlášení je tedy založena hlavně na těchto dvou operacích. Všechny

operace budou probíhat v rámci kontroly koncového systému. Kontrola by probíhala i bez nich a proto nebude do režie započítáno samotné přihlášení na koncový systém.

3.2.1.1 Kontrola klíče

Jako způsob kontroly jsem zvolil výpočet kontrolního součtu pomocí algoritmu MD5 popsaného

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.204	10.0.0.70	SSH	122	Client: Encrypted packet (len=68)
2	0.029320000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
5	0.229032000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=69 Ack=69 Win=64 Len=0
11	4.490640000	10.0.0.204	10.0.0.70	SSH	106	Client: Encrypted packet (len=52)
19	5.003986000	10.0.0.70	10.0.0.204	SSH	106	Server: Encrypted packet (len=52)
20	5.009039000	10.0.0.70	10.0.0.204	SSH	186	Server: Encrypted packet (len=132)
21	5.009226000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=121 Ack=253 Win=69 Len=0

Síťová komunikace 1

v RFC 1321. Tento algoritmus zajišťuje fixní délku přenášeného slova. Díky této vlastnosti bude kontrola jednoho souboru na koncovém systému mít vždy stejnou režii. Samotné měření proběhlo formou zachycení paketů při komunikaci přes protokol SSH. Z měření vyplynulo, že pro kontrolu jednoho klíče je potřeba přenést 750B.

3.2.1.2 Rozesílání aktuálního klíče

No.	Time	Source	Destination	Protocol	Length	Info
14	6.102720000	10.0.0.204	10.0.0.70	SSH	538	Client: Encrypted packet (len=484)
16	6.227304000	10.0.0.70	10.0.0.204	SSH	218	Server: Encrypted packet (len=164)
17	6.230599000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
18	6.230791000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=485 Ack=233 Win=63 Len=0
19	6.230954000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
20	6.231367000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
21	6.231501000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=485 Ack=369 Win=69 Len=0
22	6.231786000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
23	6.231994000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
24	6.232117000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=485 Ack=505 Win=68 Len=0
25	6.232239000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
26	6.233888000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
27	6.234088000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=485 Ack=641 Win=67 Len=0
28	6.234252000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
29	6.234731000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
30	6.234867000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=485 Ack=777 Win=67 Len=0
31	6.339861000	10.0.0.70	10.0.0.204	SSH	1170	Server: Encrypted packet (len=1116)
47	6.536898000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=485 Ack=1893 Win=69 Len=0
72	14.085623000	10.0.0.204	10.0.0.70	SSH	538	Client: Encrypted packet (len=484)
73	14.654281000	10.0.0.70	10.0.0.204	SSH	218	Server: Encrypted packet (len=164)
74	14.654995000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
75	14.655158000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=969 Ack=2125 Win=68 Len=0
76	14.655536000	10.0.0.70	10.0.0.204	SSH	138	Server: Encrypted packet (len=84)
77	14.656643000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
78	14.656805000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=969 Ack=2277 Win=67 Len=0
79	14.657168000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
80	14.659498000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
81	14.659670000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=969 Ack=2413 Win=66 Len=0
82	14.660298000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
83	14.669115000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
84	14.669224000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=969 Ack=2549 Win=66 Len=0
85	14.669544000	10.0.0.70	10.0.0.204	SSH	122	Server: Encrypted packet (len=68)
86	14.669653000	10.0.0.70	10.0.0.204	SSH	106	Server: Encrypted packet (len=52)
87	14.669710000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=969 Ack=2669 Win=65 Len=0
104	15.235890000	10.0.0.70	10.0.0.204	SSH	634	Server: Encrypted packet (len=580)
105	15.436172000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=969 Ack=3249 Win=63 Len=0
122	19.330712000	10.0.0.204	10.0.0.70	SSH	106	Client: Encrypted packet (len=52)
123	19.743806000	10.0.0.70	10.0.0.204	SSH	106	Server: Encrypted packet (len=52)
125	19.744250000	10.0.0.70	10.0.0.204	SSH	554	Server: Encrypted packet (len=500)
126	19.744402000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=1021 Ack=3801 Win=69 Len=0
128	19.745588000	10.0.0.204	10.0.0.70	SSH	138	Client: Encrypted packet (len=84)
129	19.745759000	10.0.0.204	10.0.0.70	SSH	106	Client: Encrypted packet (len=52)
134	20.083688000	10.0.0.70	10.0.0.204	SSH	90	Server: Encrypted packet (len=36)
135	20.107281000	10.0.0.70	10.0.0.204	TCP	54	22→57299 [ACK] Seq=3837 Ack=1157 Win=397 Len=0
138	20.283212000	10.0.0.204	10.0.0.70	TCP	54	57299→22 [ACK] Seq=1157 Ack=3837 Win=68 Len=0

Síťová komunikace 2

Klíče bude daemon zapisovat do souboru pomocí příkazu „echo“, jehož výstup bude přesměrován pomocí příkazu „tee“. Pro nahrání dvou veřejných částí klíčů o velikost 2048 bitů je potřeba přenést 7422B. ≈ 7450B

3.2.2 Centrální databáze

Pro účely testování režie přihlašování za použití architektury Centrální databáze byl použit jeden

No.	Time	Source	Destination	Protocol	Length	Info
47	2.151325000	10.0.0.105	10.0.0.106	TCP	74	33312→389 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=4294778224 TSecr=0 WS=64
48	2.151504000	10.0.0.106	10.0.0.105	TCP	74	389→33312 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1140737 TSecr=42
49	2.151789000	10.0.0.105	10.0.0.106	TCP	66	33312→389 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=4294778225 TSecr=1140737
50	2.151947000	10.0.0.105	10.0.0.106	LDAP	128	bindRequest(1) "cn=nssproxy,ou=users,dc=company,dc=com" simple
51	2.152013000	10.0.0.106	10.0.0.105	TCP	66	389→33312 [ACK] Seq=1 Ack=63 Win=14528 Len=0 TSval=1140737 TSecr=4294778225
52	2.152485000	10.0.0.106	10.0.0.105	LDAP	80	bindResponse(1) success
53	2.152719000	10.0.0.105	10.0.0.106	TCP	66	33312→389 [ACK] Seq=63 Ack=15 Win=14656 Len=0 TSval=4294778226 TSecr=1140737
54	2.152986000	10.0.0.105	10.0.0.106	LDAP	213	searchRequest(2) "ou=users,dc=company,dc=com" wholeSubtree
55	2.153271000	10.0.0.106	10.0.0.105	LDAP	544	searchResEntry(2) "cn=test.user,ou=users,dc=company,dc=com"
56	2.153393000	10.0.0.106	10.0.0.105	LDAP	80	searchResDone(2) success [1 result]
57	2.153714000	10.0.0.105	10.0.0.106	TCP	66	33312→389 [ACK] Seq=210 Ack=507 Win=15680 Len=0 TSval=4294778227 TSecr=1140738
58	2.153819000	10.0.0.105	10.0.0.106	LDAP	73	unbindRequest(3)
59	2.153932000	10.0.0.106	10.0.0.105	TCP	66	389→33312 [FIN, ACK] Seq=507 Ack=217 Win=15552 Len=0 TSval=1140739 TSecr=4294778227
60	2.153940000	10.0.0.105	10.0.0.106	TCP	66	33312→389 [FIN, ACK] Seq=217 Ack=507 Win=15680 Len=0 TSval=4294778227 TSecr=1140738
61	2.153999000	10.0.0.106	10.0.0.105	TCP	66	389→33312 [ACK] Seq=508 Ack=218 Win=15552 Len=0 TSval=1140739 TSecr=4294778227
62	2.154267000	10.0.0.105	10.0.0.106	TCP	66	33312→389 [ACK] Seq=218 Ack=508 Win=15680 Len=0 TSval=4294778228 TSecr=1140739
65	2.208413000	10.0.0.105	10.0.0.106	TCP	74	33313→389 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=4294778281 TSecr=0 WS=64
66	2.208582000	10.0.0.106	10.0.0.105	TCP	74	389→33313 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1460 SACK_PERM=1 TSval=1140794 TSecr=42
67	2.208855000	10.0.0.105	10.0.0.106	TCP	66	33313→389 [ACK] Seq=1 Ack=1 Win=14656 Len=0 TSval=4294778282 TSecr=1140794
68	2.209022000	10.0.0.105	10.0.0.106	LDAP	128	bindRequest(1) "cn=nssproxy,ou=users,dc=company,dc=com" simple
69	2.209335000	10.0.0.106	10.0.0.105	TCP	66	389→33313 [ACK] Seq=1 Ack=63 Win=14528 Len=0 TSval=1140795 TSecr=4294778282
70	2.209501000	10.0.0.106	10.0.0.105	LDAP	80	bindResponse(1) success
71	2.209717000	10.0.0.105	10.0.0.106	TCP	66	33313→389 [ACK] Seq=63 Ack=15 Win=14656 Len=0 TSval=4294778283 TSecr=1140795
72	2.210009000	10.0.0.105	10.0.0.106	LDAP	213	searchRequest(2) "ou=users,dc=company,dc=com" wholeSubtree
74	2.210244000	10.0.0.106	10.0.0.105	LDAP	544	searchResEntry(2) "cn=test.user,ou=users,dc=company,dc=com"
75	2.210336000	10.0.0.106	10.0.0.105	LDAP	80	searchResDone(2) success [1 result]
76	2.210689000	10.0.0.105	10.0.0.106	TCP	66	33313→389 [ACK] Seq=210 Ack=507 Win=15680 Len=0 TSval=4294778284 TSecr=1140795
77	2.210812000	10.0.0.105	10.0.0.106	LDAP	73	unbindRequest(3)

Síťová komunikace 3

server, na kterém jsou data ukládána a načítána za pomoci LDAP protokolu a druhý server, který své klíče na tomto serveru vyhledává. Klíče byli ukládány podle schématu „openssh-lpk-openldap.schema”. Komunikace mezi daemon a serverem s klíči není součástí režie.

Pro úspěšnou autentizaci je nutné přenést 4380B ≈ 4400B.

3.2.3 Srovnání

Následující graf ukazuje, kolikrát by se uživatel musel na koncový systém přihlásit, aby se vyplatila architektura Centrálního výdeje tokenů. Výpočet předpokládá, že jeden uživatel má přístup na sto koncových systému a nový klíč se generuje (má platnost) jednou za 1, 7 nebo 30 dnů. Vzorec použitý při výpočtu:

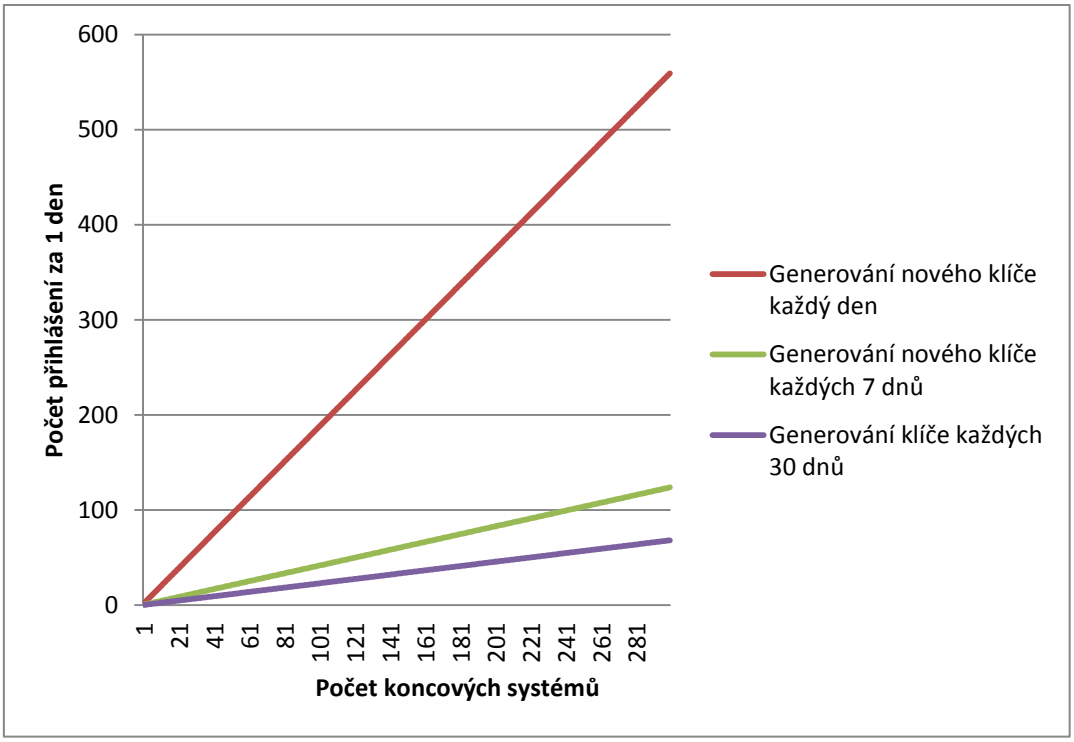
$$\frac{(p \times K + N) \times k}{p \times L}$$

kde:

- p = Platnost klíče ve dnech.
- k = Počet koncových systémů.
- K = Počet bajtů potřebných ke kontrole jednoho klíče.
- N = Počet bajtů potřebných k nahrání jednoho klíče.
- L = Počet bajtů potřebných k přihlášení přes Centrální databázi.

Příklad výpočtu pro jednoho uživatele, s přístupem na sto koncových systémů, při změně klíče každý den:

$$\frac{(1 \times 750 \ + \ 7450) \times 100}{1 \times 4400} \approx 187$$



Graf počtu přihlášení 1

Z grafu vyplívá, že pokud má jeden uživatel přístup na sto koncových systémů a klíč by se měnil každý den, musel by se každý den přihlásit alespoň 187krát, aby se vyplatilo rozesílat klíče na koncové systémy. Pokud by se klíč měnil jednou měsíčně, musel by se uživatel přihlásit jen 23krát.

3.3 Požadavky na aplikaci

Z analýzy vyplynulo několik požadavků, které musí návrh splňovat, aby byl konkurenceschopný.

3.3.1 Nefunkční požadavky

1. Aplikace musí spravovat identity (musí být schopná vytvářet, mazat, upravovat účty na koncových systémech).
2. Uživatelé se na koncové stanice nesmějí přihlašovat pomocí hesel.
3. Aplikace musí umožňovat SSO pro uživatele koncových systémů.
4. Do aplikace musí být možno připojit jakýkoliv unixový systém, který podporuje správu identit a přihlašování jinak než heslem.
5. Aplikaci budou smět používat uživatelé nezávisle na operačním systému, který na svých pracovních stanicích používají.
6. Aplikace bude schopna detekovat změny v nastavení účtů na připojených systémech a tyto změny i opravit.
7. Nebude nutné výrazně upravovat připojované systémy.
8. Uživatelé aplikace nebudou muset výrazně měnit své pracovní návyky.

9. Aplikace bude připojitelná k existujícím řešením správy identit.
10. Aplikace bude umožňovat přihlašování i při své krátkodobé nedostupnosti.
11. Aplikaci půjde užívat pomocí webového rozhraní.

3.3.2 Funkční požadavky

1. Aplikace bude podporovat přihlášení administrátora aplikace.
2. Administrátor bude moci:
    a. Vytvářet uživatele.
    b. Mazat uživatele.
    c. Přidělovat uživatelům koncové systémy.
    d. Odebírat uživatelům koncové systémy.
    e. Připojovat koncové systémy.
    f. Odebírat koncové systémy.
    g. Nastavovat konfigurační soubory aplikace.
    h. Vytvářet jiné administrátory.
3. Aplikace bude podporovat přihlášení uživatele.
4. Uživatel v aplikaci bude moci:
    a. Listovat nastavení svého účtu.
    b. Získat na omezenou dobu přístup do jemu přiřazených koncových systémů.

4 Popis aplikace

4.1 Použití dvou klíčů

Pro přihlašování uživatelů na koncové systémy jsem zvolil přihlašování pomocí veřejných klíčů - Public Key Cryptografhy Standards (dále jen PKCS) definované v RFC 3447. Důvodem byly:

- Široká podpora standardů na Unix/Linux platformách.
- Možnost používání klíčů k přihlášení bez nutnosti zadávání jiných autentizačních údajů.
- Bezpečnost standardů.

Samotný proces výměny šifrovacího klíče je zabezpečen asymetrickým šifrování popsaným v RFC 3447. Zůstává ovšem problém prokázání identity vlastníka klíče. Při dnešním způsobu používání PKCS se jedna komunikující strana (ta s veřejnou částí klíče) musí spolehnout na to, že druhá komunikující strana (vlastník soukromého klíče) zajistila zabezpečení použitého klíče. V realitě to znamená, že pokud administrátor vloží veřejnou část klíče uživatele na svůj server, vkládá tím do uživatele značnou důvěru. Přesto, že tento klíč slouží k zabezpečení komunikace, administrátor serveru nemá jak účinně zajistit, aby uživatel měl klíč chráněný passphrase, aby nenechával svůj klíč na jiných médiích než je jeho disk, aby neumožňoval používání svého počítače třetím osobám apod.

V prostředí internetu se používá architektura PKI. Zde je definován pojem certifikát typu X.509. Ten řeší prokázání identity vlastníka klíče pomocí podepsání klíče důvěryhodnou autoritou a specifikování způsobu jeho použití, vytvoření certifikátu. Tento přístup bohužel neřeší problém odcizení klíče od právoplatného vlastníka. Zajištění bezpečnosti klíče si zajišťuje vlastník klíče sám. Specifikace způsobu použití ale značně omezuje možnosti použití klíče po jeho odcizení. Příkladem takové specifikace je vystavení klíče na konkrétní jméno domény. Bohužel jsem nenašel žádný atribut, který by jednoznačně identifikoval daného uživatele, aniž by značně omezil možnosti používání klíče. Z tohoto důvodu aplikace nebude využívat PKI ale jen jeho podmnožinu PKCS.

Prvním návrhem na zabránění odcizení klíče bylo odebrání klíče od uživatele. Uživatel by dostal klíč zapůjčení jen na dobu potřebnou k autentizaci. Pro ztížení odcizení klíče by se klíč držel jen v RAM paměti a po vypnutí nebo restartu počítače by došlo k jeho ztrátě. Toto řešení ale značně komplikuje nepopiratelnost akcí uživatele (nepopiratelností se zabývá například i Česká technická norma ČSN ISO/IEC 13888). Jedná se o případ, kdy by uživatel popřel, že daný klíč použil. Protože

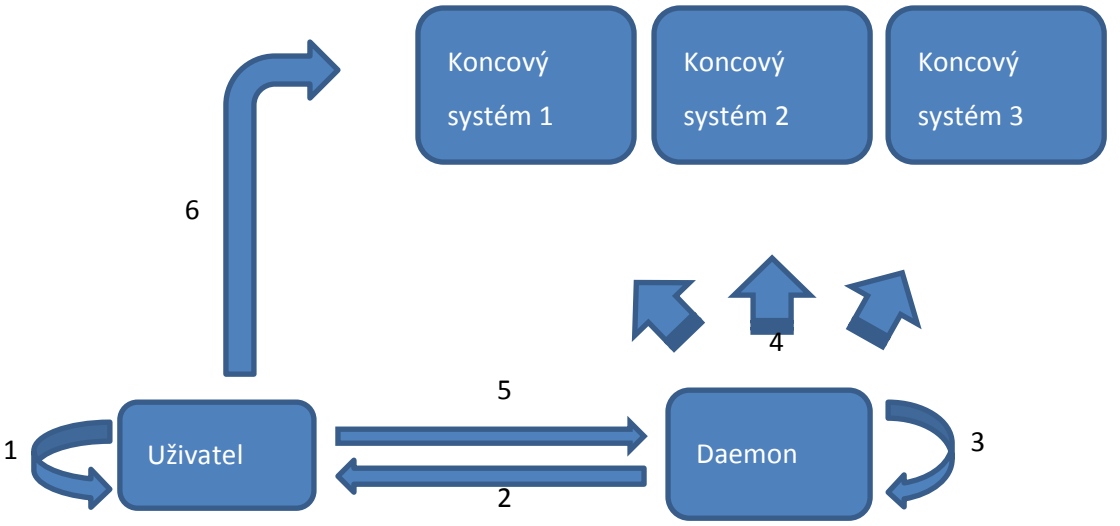
veškeré údaje potřebné k autentizaci na daný systém by vlastnila aplikace generující zapůjčovaný klíč, bylo by nutné jednoznačně dokázat, že klíč nemohl být nikým zneužit. Naproti tomu, pokud by si klíč uživatel vygeneroval sám, byl by za jeho zabezpečení plně odpovědný jedině on.

Řešením obou problémů, jak problému odcizení klíče uživateli, tak problému nepopiratelnosti uživatelovi autentizace, je kombinace obou výše zmíněných řešení. Aby se uživatel mohl přihlásit, musí vlastnit dva klíče. Jeden klíč si generuje sám a jeho privátní část se nikdy nedostane na server aplikace. O zabezpečení tohoto klíče se uživatel stará sám. Druhý klíč si uživatel stahuje ze serveru aplikace. Aby byl uživateli klíč vydán, musí se uživatel autentizovat a tím prokázat svoji identitu. O zabezpečení druhého klíče se stará aplikace sama (může jej měnit v nastavených intervalech). Uživatel bude mít klíče uloženy v RAM paměti a tím značně zvýší složitost jejich odcizení.

4.2 Základní princip použití

Aplikace bude umožňovat přístup na Unixové stroje pomocí distribuce SSH klíčů. K tomu, aby mohl uživatel používat aplikaci, musí mít nainstalovaný a spuštění SSH Agent[1]. Uživatel si nejdříve vygeneruje vlastní SSH klíč. Jeho privátní část nahraje do SSH Agenta. Při prvním přihlášení k daemonovi aplikace nahraje uživatel veřejnou část SSH klíče do daemona. Dále si nechá daemonem vygenerovat druhý klíč. Aplikace se spojí s SSH agentem na uživatelově počítači a zašle do něj privátní část. V tuto chvíli má uživatel v SSH Agentovi dva privátní klíče. Daemon aplikace rozešle oba veřejné klíče na servery, kam má uživatel mít přístup. Všechny servery pod správou aplikace budou umožňovat jediný druh přístupu a to pomocí dvou SSH klíčů. Jakmile se uživatel pokusí na daný server přihlásit, SSH Agent při autentizaci vyzkouší všechny klíče, které má v sobě nahrané a uživateli bude přístup umožněn.

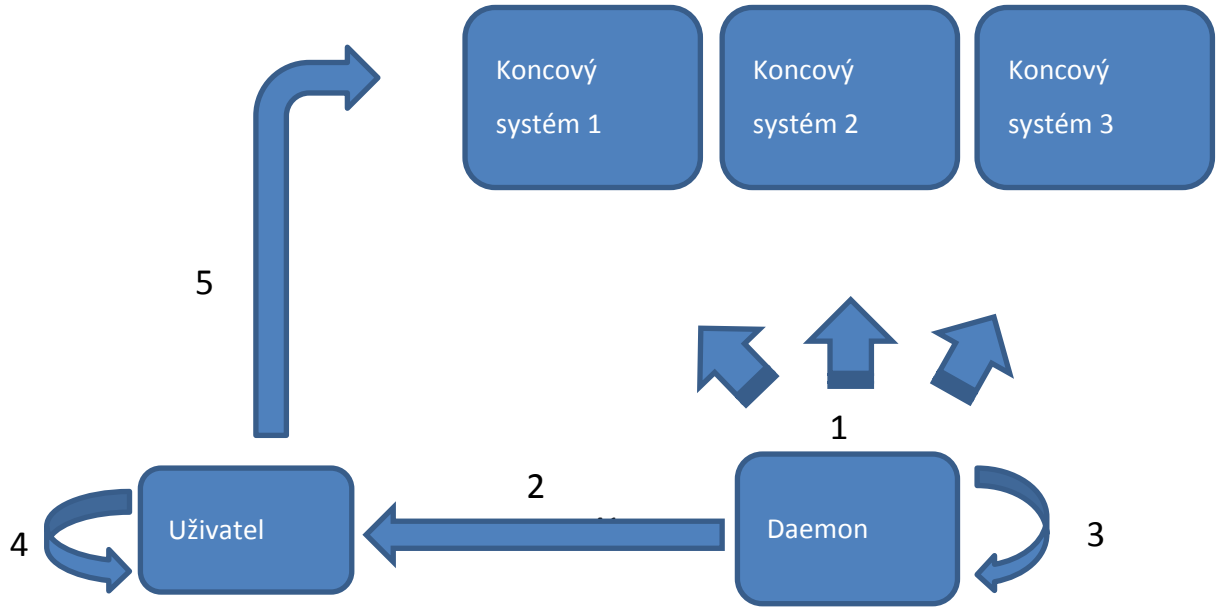
[1] Například PageAnt od firmy PuTTY, stažitelný na stránkách http://www.putty.org/



Popis komunikace 1

Obrázek popisuje kroky nutné ke stažení klíče při prvním přihlášení:

1. Uživatel si na své pracovní stanici vygeneruje veřejnou i privátní část klíče. Privátní část si načte do SSH Agenta.
2. Uživatel zašle veřejnou část klíče deamonovi aplikace. S veřejnou částí klíče zašle i žádost o vygenerování nového klíče a passphrase, kterou má být klíč zašifrován. Ke komunikaci s daemonem aplikace bude uživatel využívat klienta aplikace.
3. Daemon aplikace vygeneruje veřejnou a privátní část klíče a zašifruje jej pomocí zadané passphrase. Bude vygenerován také jeden pár, který se uloží jako záloha do databáze.
4. Daemon aplikace rozešle obě veřejné části klíčů na uživateli přiřazené koncové systémy.
5. Uživatel si stáhne privátní část daemonem vygenerovaného klíče, rozšifruje ji pomocí passphrase a vloží si ji do SSH Agenta.
6. Nyní má uživatel dva privátní části klíče, pomocí kterých může přistoupit na jakýkoliv jemu přiřazený koncový systém.



Popis komunikace 2

Při následném běžném používání aplikace se postup zjednodušuje

1. Po vypršení nastaveného času od posledního stažení klíče uživatelem daemon aplikace nahradí veřejnou část klíče, který sám vygeneroval, veřejnou částí klíče, který má v záloze.
2. Uživatel si stáhne klíč z daemona aplikace, který si rozšifrovaný uloží do SSH Agenta. Aby mu to daemon aplikace umožni, bude po něm požadovat passphrase ke klíči.
5
3. Daemon aplikace vygeneruje náhradní privátní a veřejnou část klíče, zašifruje je pomocí zadané passphrase a uloží si je do databáze.
4. Uživatel si vloží do SSH Agenta privátní část vlastního klíče.
5. Nyní má uživatel dvě privátní části klíčů, pomocí kterých může přistoupit na jakýkoliv jemu přiřazený koncový systém.

4.3 Přístup uživatele

Uživatel se pomocí agenta aplikace připojí k daemonu aplikace. Způsob, kterým se vůči daemonu autentizuje, může být volitelný (RSA, heslo nebo více hesel). Protože je toto jediné místo, kde se ověřuje identita uživatele, měla by mít autentizace více faktorů. Po přihlášení k daemonu aplikace si bude moci uživatel nechat vygenerovat klíč a stáhnout si ho do SSH Agenta. Daemon si vyžádá od uživatele passphrase, se kterou se klíč při generování zašifruje. Klíč se uživateli stáhne do SSH Agenta, tedy do RAM paměti. Uživatel se pak pomocí tohoto klíče přihlašuje na všechny servery. Protože je klíč v SSH Agentovi, není při přihlašování na servery po uživateli vyžadována žádná autentizace, jen jméno (SSO řešení).

4.4 SSH agent

Klíče si uživatel stahuje do takzvaného SSH Agenta. Jedná se o program, který drží klíče uživatele v RAM paměti. Při restartu nebo vypnutí počítače je klíč ztracen. Klíč je v SSH Agentovi držen v rozšifrované formě a uživatel se s jeho pomocí přihlašuje na ostatní servery.

4.5 Práce daemona aplikace

Na serveru, kde běží daemon aplikace, jsou uloženy všechny klíče uživatelů. Daemon generuje pro uživatele více jak jeden klíč. Prací daemona je zajistit, aby se klíč nacházel na všech serverech, kam má uživatel přístup. Dále daemon pravidelnou (pravděpodobně jednodenní) kontrolou zajišťuje, že na serveru nevznikly nové účty, a že stávající účty mají nastaveno správné oprávnění.

4.6 Session uživatele

Daemon aplikace kontroluje dobu od posledního stažení klíče a porovnává ji s povolenou platností session. Pokud dojde k překročení nastaveného času, odebere ze všech cílových serverů klíč a tím zamezí uživateli k těmto serverům přístup. Aktivní připojení uživatel s koncovým systémem se tímto způsobem neukončí. Pokud se uživatel bude chtít přihlásit znovu, nové přihlášení mu již nebude umožněno. Hlavním důvodem tohoto chování je, že by násilné ukončení uživatelova spojení mohlo mít neblahé následky na jeho práci. K tomu, aby uživatel mohl vytvořit nové spojení s koncovým systémem, si musí znovu stáhnout klíč.

4.7 Výměna klíčů

Daemon aplikace pro uživatele generuje více, jak jeden klíč. Na server je jich také nahráno více a to z důvodu omezení zátěže při jejich rozesílání. Jeden klíč uživatele je platný po dobu session uživatele od jeho posledního stažení. Například: pokud je session nastavena na jeden den, jsou v noci na server nahrány dva klíče. Uživatel se ráno připojí k daemonu aplikace a jeden klíč si stáhne. Pokud se ten samý den připojí k daemonu znovu, stáhne se stejný klíč jako ráno. Po uplynutí jednoho dne odebere daemon daný klíč ze serveru a připraví do zásoby nový. Uživatel se druhý den ráno připojí k daemonu a stáhne si (pomocí stejné passphrase) nový klíč, který bude používat celý následující den.

4.8 Vlastnosti

V následujícím textu se pokusím stručně shrnout hlavní výhody a nevýhody tohoto řešení oproti již existujícím aplikacím.

4.8.1 Výhody

4.8.1.1 Kritický bod

- Při krátkodobé nedostupnosti aplikace není ohrožena práce uživatelů. Většina uživatelů má stažené klíče v agentovi a k další práci distribuční server nepotřebuje.
- Při dlouhodobé nedostupnosti se mohou klíče rozeslat (jsou zašifrované a proto nehrozí jejich zneužití) a překlenout tak problémové období.

4.8.1.2 Hardwarové nároky

- Snížené nároky na infrastrukturu. Uživatelé sice musí přistupovat pravidelně k daemonu aplikace, daemon ale nemusí udržovat otevřená spojení s ostatními servery (případ

architektury „man in the middle“), ani jim pravidelně odpovídat z důvodů autentizace (případ architektury „centrální databáze“).

- Celá architektura je založena na jednom serveru (nebo více v clusteru), které nemusejí být extra silné (většina úloh běží v noci a pracuje se výhradně s ssh).

4.8.1.3 Používání hesel

- Při kompromitaci DB daemona aplikace se útočník dostane jen ke klíčům, které jsou zašifrované pomocí passphrase a mají platnost jednoho dne. Nikde nejsou žádná hesla.
- Klíče si uživatelé stahují do RAM paměti odkud je velice složité je dostat.

4.8.1.4 Cena

- Cena takového řešení by byla řádově nižší s porovnáním s existujícími aplikacemi.
- Nejsou potřeba investice do infrastruktury (servery nebo síť).
- Jsou potřeba jen minimální úpravy na připojovaných serverech.

4.8.2 Nevýhody

- Architektura nepodporuje online monitorování činnosti uživatele, jako je možné v případě architektury „man in the middle“. Upravení architektury tak, aby z části podporovala „man in the middle“ není úplně nemožné.
- Uživatelé musejí mít nainstalovaného SSH Agenta na svých pracovních stanicích.

4.9 Komponenty návrhu

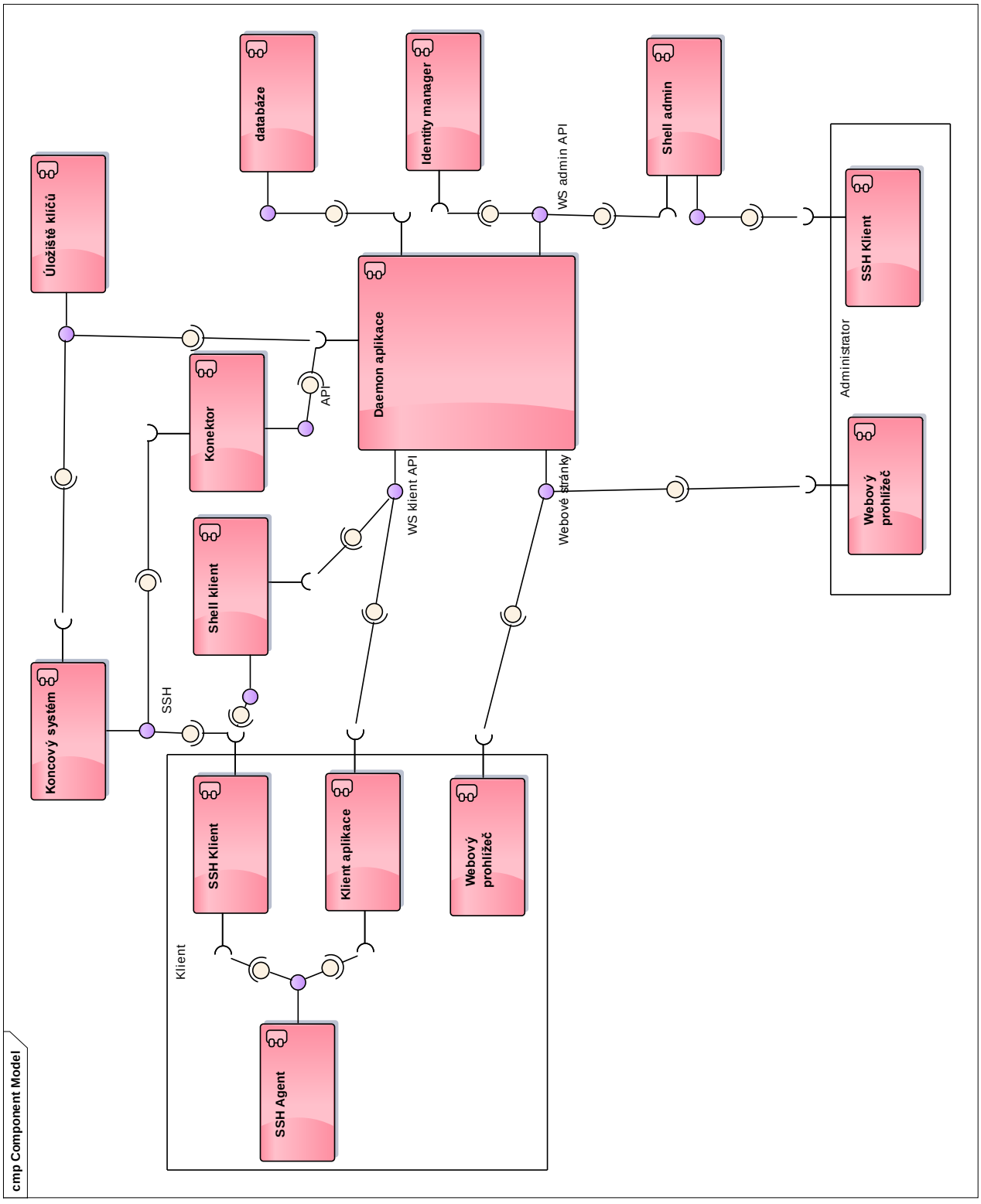


Diagram komponent 1

4.9.1 Popis component

4.9.1.1 SSH klient

SSH klient je aplikace instalovaná na pracovní stanice uživatelů nebo administrátorů. Zprostředkovává připojení a vlastní spojení pomocí protokolu SSH. SSH klientů je velké množství. Asi nejznámějším a nejpoužívanějším je „PuTTY“. Mezi další patří například: „JuiceSSH“, „DameWare“ nebo „SunSSH“. Pro používání tlustého klienta bude ze začátku certifikováno jen „PuTTY“ a to z důvodu otevřeného API přes příkazový řádek.

4.9.1.2 SSH Agent

Pro ochranu klíčů před odcizením podporuje protokol RSA i DSA možnost zašifrování klíče symetrickou šifrou. Její klíč se nazývá „passphrase“. Pokud je klíč zašifrován, je nutné jej rozšifrovat při každém použití. To je samozřejmě velice nepříjemné a proto velké množství uživatelů klíče vůbec nešifruje. Jako odpovědí na tento problém vznikly takzvané SSH Agenty, které daný klíč rozšifrují jen jednou a rozšifrovaný jej drží v RAM paměti počítače. Při restartu nebo vypnutí počítače je rozšifrovaný klíč ztracen, ale v rámci jednoho běhu je možné rozšifrovaný klíč používat a po uživateli není požadována pokaždé passphrase.

4.9.1.3 Shell klient

Klientský shell se zobrazí uživateli po přihlášení do aplikace pomocí SSH protokolu. Bude se jednat o skript, který bude nastaven uživateli jako shell v souboru „/etc/passwd“. Po přihlášení shell nabídne uživateli jen omezený počet možností:

1. Stáhnout klíč
2. Vygenerovat klíč
3. Nahrát veřejný klíč
4. Vypsat celý veřejný klíč
5. Rozeslat klíč
6. Vypsat vlastněné účty

Shell nebude nabízet plnohodnotnou práci s aplikací, ale bude dostatečný pro základní aplikační případ užití. Uživatel si s jeho pomocí dokáže nastavit celý klíč a nechat jej rozeslat na připojené služby. Dokáže si také pomocí shellu klíč stáhnout a na koncové služby se připojit.

Shell bude ve své podstatě Tlustý klient, který bude omezen vyjadřovacími schopnostmi příkazového řádku. Bude komunikovat se stejnou webovou službou jako Tlustý klient na straně klienta.

4.9.1.4 Shell admin

Administrátorům bude také umožněno pracování s aplikací přes SSH protokol. Na rozdíl od uživatelů jim bude zobrazen klasický shell serveru (bash, ksh, sh). Aplikace samotná bude ovládána pomocí implementovaných příkazů. Každý příkaz bude implementace jedné metody webové služby.

4.9.1.5 Databáze

Databáze obsahuje informace potřebné k běhu aplikace. Jsou zde uloženy účty, atributy, koncové systémy, konektory a veškeré nastavení. Pomocí relací mezi tabulkami je realizována většina datových závislostí aplikace.

Detailní popis databáze se nachází v kapitole 4.11.

4.9.1.6 Webový prohlížeč

Aplikace bude podporovat přístup pomocí webových stránek. Budou existovat dvě adresy. Jedna pro uživatele a jedna pro administrátory. Pro Administrátory budou webové stránky poskytovat plnohodnotný přístup k aplikaci a budou moci plánovat, spouštět i provádět všechny úpravy v aplikaci.

Pro uživatele bude přístup přes webové stránky spíše informativní. Pomocí technologie webových stránek se dají velice snadno a přehledně zobrazovat uživatelova data grafickou formou. Uživatel si bude moci změnit veřejnou část osobního klíče, ale nebude si moci stáhnout klíč privátní.

4.9.1.7 Konektor

Konektory jsou popsány v kapitole 4.12.2.

4.9.1.8 Koncový system

Koncové systémy jsou popsány v kapitole 4.11.2.1.

4.9.1.9 Identity Manager

Komponentou Identity Manager je myšlena jakýkoliv aplikace pro správu identit. Aplikace bude mít webovou službu připravenou pro administrace aplikačních identit. Díky tomu bude možnost správu aplikace z velké části přenechat na zvoleném produktu pro správu Identit. Tím se velice zjednoduší práce administrátora.

4.9.1.10 Tlustý klient

Tlustý klient (dále už jen klient) je program, který zprostředkovává komunikaci s daemonem aplikace pomocí rozhraní webových služeb. Program je nutné nainstalovat na pracovní stanici

uživatele. Tlustý klient komunikuje s SSH Agentem a SSH klientem, které se musí na pracovní stanici uživatele také nacházet. Pomocí klienta bude uživatel schopný provádět všechny operace, které aplikace podporuje. Návrh klienta je popsán v kapitole 4.13.3.

4.9.1.11 Úložiště klíčů

Aplikace bude podporovat ze začátku dva typy ukládání klíčů. Návrh ale bude umožňovat přidání dalších typů v budoucích verzích aplikace.

4.9.1.11.1 Klasické ukládání

Jako klasické ukládání je považováno ukládání klíče přímo na připojený koncový systém. O práci s klíčem se stará konektor. Tato možnost musí být konektorem podporována. Klíče jsou ukládány do souborů, jejichž jméno a umístění je závislé na nastavení SSHD (nastavení v souboru sshd_config). Výhodou tohoto typu je absence centrálního prvku. Po distribuci klíčů si samotnou autentizaci řeší koncový systém sám a není závislý na žádném dalším systému. Nevýhodou je vyšší režie potřebná k distribuci klíčů.

4.9.1.11.2 LDAP

Nastavení tohoto typu úložiště klíčů způsobí, že se klíče uživatelů nebudou distribuovat na koncový systém, ale na specifikovaný systém za pomoci protokolu LDAP. Mapování mezi účtem uživatele na koncovém systému a klíčem v struktuře LDAPu bude pomocí jména účtu. Samotný klíč bude uložen podle schématu „openssh-lpk-openldap.schema" (instalovaného v rámci balíků openssh-ldap).

Pro nastavení koncového systému tak, aby klíče uživatelů vyhledával na jiném serveru pomocí protokolu LDAP je nutná konfigurace SSH daemona a instalace balíku „openssh−ldap". Do souboru „sshd_config" je nutné přidat následující řádky:

AuthorizedKeysCommand /usr/libexec/openssh/ssh-ldap-wrapper
AuthorizedKeysCommandRunAs nobody
PubkeyAuthentication yes

Pro úspěšnou autentizaci je také nutné vytvořit konfigurační soubor, jenž bude ssh-ldap-wrapper využívat. Ten se musí nacházet v /etc/ssh/ldap.conf a vypadat asi následovně:

```
# /etc/ssh/ldap.conf
#
# OpenLDAP client configuration.
# Do NOT confuse this file with /etc/ldap.conf.
# See ldap.conf(5) for details.
```

BASE **ou=users,dc=example,dc=org**

URI **ldap://ldap1.example.org**

BINDDN **cn=nssproxy,ou=users,dc=example,dc=org**

BINDPW **Change.Me!**

TLS_CACERT **/etc/pki/tls/certs/rootca.crt**

TLS_REQCERT **allow**

TIMELIMIT **15**

TIMEOUT **20**

EOF

Pokud je vyžadováno použití protokolu SSL (protože heslo a jméno uživatele je zasíláno na server jako text, nepoužití SSL lze považovat za značný bezpečnostní problém), je nutné umístit na koncový systém i CRT certifikát serveru, aby mu SSHD daemon důvěřoval.

4.10 Rozhraní daemona aplikace

Jak je z popisu komponent patrná, uživatel i administrátor budou mít více způsobů jak aplikaci používat.

4.10.1 Webové služby

4.10.1.1 Uživatelské služby

Webová služba pro uživatele zprostředkovává všechny potřebné metody k tomu, aby uživatel mohl aplikaci plně využívat. Služba bude ke komunikaci používat protokol SOAP a poskytovat následující metody:

- ping()

Metoda pro otestování správného spojení s daemonem aplikace.

- List<String> downloadPrivateKey(String type)

Uživatel si takto bude moci stáhnout klíč. Pomocí parametru „type“ může specifikovat, do jakého formátu má být klíč před zasláním změněn.

- List<String> getServerStatus()

Jediný způsob, jak může daemon aplikace upozornit klienty aplikace na nestandardní situaci (nutnost okamžitého přegenerování klíče apod.). Agenti budou pravidelně kontrolovat status daemona aplikace a následně reagovat na nastalé situace.

- List<String> getUserEndpoints()

Metoda vrátí seznam koncových systémů, které má uživatel přiřazeny.

- generateNewPrivateKey(char[] passphrasse)

Pokud uživatel potřebuje okamžitě přegenerovat klíč, nebo jej poprvé vytvořit (první přihlášení), použije tuto metodu.

- setUserPublicKey(List<String> key, String type)

Metoda slouží pro nastavení veřejné části klíče, kterou pak daemon aplikace rozesílá na uživateli přiřazené koncové systémy.

- startSynchronization()

Vynucení rozeslání klíče na koncové systémy.

- List<String> getUserDataForEndpoint(String name)

Umožnění detailnějšího náhledu na atributy uživatele, které jsou nastaveny na koncovém systému.

- List<String> getKnownHost()

Metoda pro získání aktuálního seznamu daemonem aplikace zpravovaných serverových klíčů.

4.10.1.2 Administrátorské služby

Administrátoři aplikace budou moci ovládat daemona aplikace přes webové rozhraní, nebo přes příkazový řádek. Příkazy z příkazového řádku budou komunikovat se stejnou webovou službou jako případné konektory pro Identity manager. Oběma způsoby bude umožněno kompletní ovládání daemona aplikace. Služba bude podporovat následující metody, které jsou rozřazeny do dvou logických celků:

4.10.1.2.1 Metody pro manipulaci s daty

- List<String> listUsers()
- createUser(Map<String,String>)
- deleteUser(String name)
- editUser(Map<String,String>)
- getUser(String name)
- List<String> listUserAttributes()
- createUserAttribute(Map<String,String>)
- deleteUserAttribute(String name)

- editUserAttribute(Map<String,String>)
- getUserAttribute(String name)
- List<String> listMirrorAccounts()
- createMirrorAccount(Map<String,String>)
- deleteMirrorAccount(String name)
- editMirrorAccount(Map<String,String>)
- getMirrorAccount(String name)
- List<String> listEndpointAttributes()
- createEndpointAttribute(Map<String,String>)
- deleteEndpointAttribute(String name)
- editEndpointAttribute(Map<String,String>)
- getEndpointAttribute(String name)
- List<String> listEndpoints()
- createEndpoint(Map<String,String>)
- deleteEndpoint(String name)
- editEndpoint(Map<String,String>)
- getEndpoint(String name)
- List<String> listConnectors()
- createConnector(Map<String,String>)
- deleteConnector(String name)
- editConnector(Map<String,String>)
- getConnector(String name)

Z důvodu jednodušší údržby mají metody jako vstup mapu názvů a hodnot. Tato architektura je zvolena proto, aby nebylo nutné dělat úpravy na všech klientech aplikace při každé drobné změně rozhraní (například přidání nebo ubrání atributu).

4.10.1.2.2 Metody pro spouštění úloh

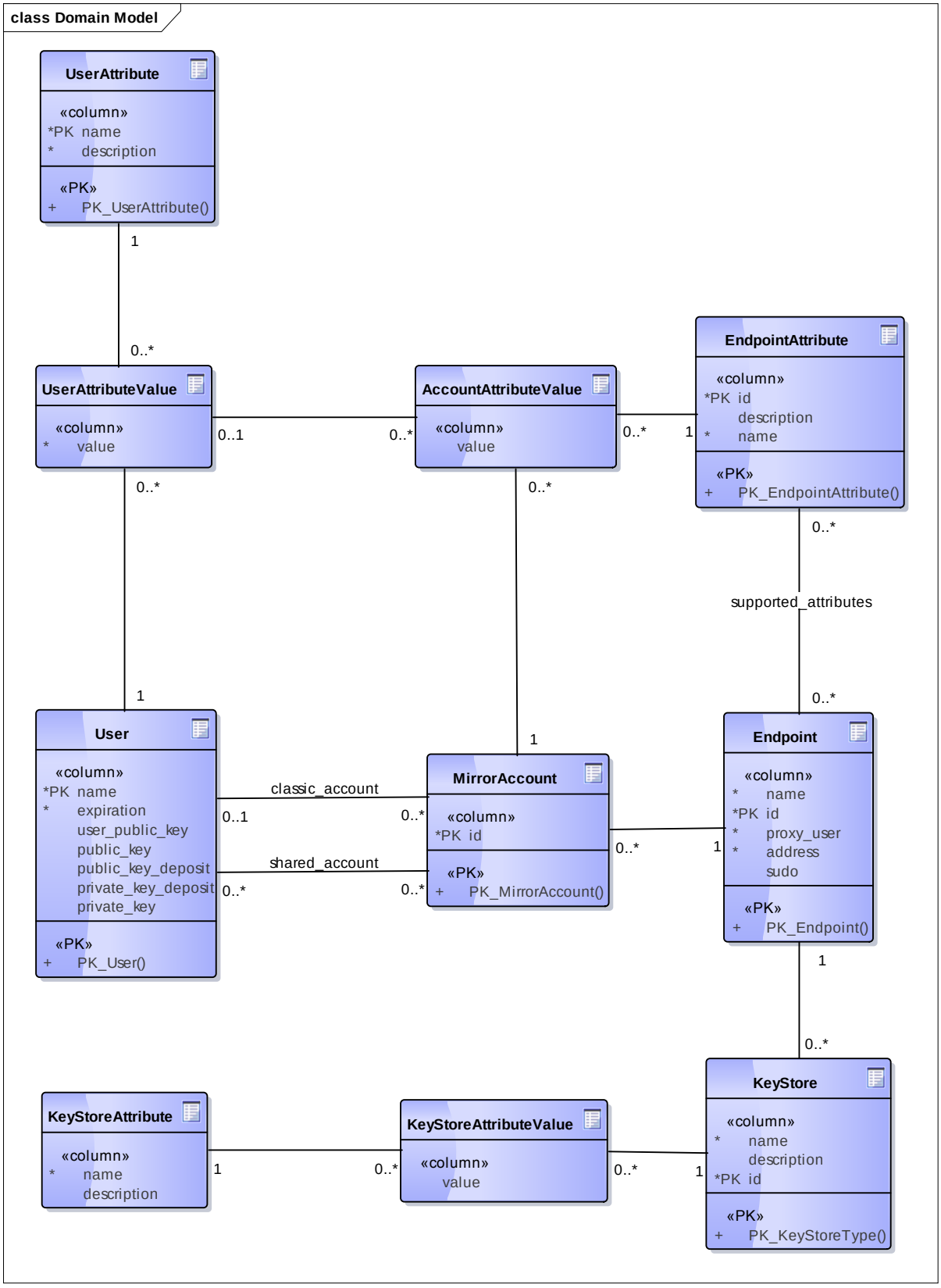
Úlohy pro administraci jsou popsány v kapitole 4.15. Zde budou jen popsány způsoby jejich spuštění.

- synchronizeAccounts(List<String>, List<String>)
- Reconcile(List<String>)
- changeKey(List<String>)
- checkKeys(List<String>, List<String>)

- searchForSimilar(List<String>, List<String>)

Prvním vstupem procesů je vždy seznam koncových systémů, na kterých má proces proběhnout. Druhým vstupem je u procesů „checkKeys“ a „synchronizeAccounts“ seznam Zrcadlových účtů, kterými se má proces zabývat. Proces „searchForSimilar“ očekává na druhém vstupu seznam atributů, podle kterých má párování proběhnout. Pokud je jeden ze vstupů prázdný, bude proces proveden na všech dostupných koncových systémech nebo Zrcadlových účtech. V případě prázdného vstupu atributů se bude k párování využívat jako jediný atribut jméno uživatele.

4.11 Databáze



Domain diagram 1

4.11.1 Popis tabulek

Datový model aplikace se skládá z těchto tabulek.

- User

Tabulka uživatelů aplikace. Každý, administrátor nebo uživatel, musí být v tabulce založen. V tabulce jsou základní atributy uživatelů.

Název atributu	Popis
Expiration	Datum, do kterého je účet platný. Po jeho přesažení aplikace uživatele automaticky zneplatní.
Name	Jméno účtu v aplikaci. Slouží také jako přihlašovací jméno.
Id	Unikátní identifikátor účtu v rámci celé aplikace.
user_public_key	Veřejná část klíče generovaného uživatelem aplikace. Uživatel veřejnou část nahraje po prvním přihlášení.
public_key	Veřejná část klíče generovaného aplikací.
public_key_deposit	Veřejná část náhradního klíče, generovaného aplikací.
private_key	Soukromá část klíče generovaného aplikací.
private_key_deposit	Soukromá část náhradního klíče, generovaného aplikací
last_download_key	Čas posledního stažení klíče z aplikace.

Databázová tabulka 1

- UserAttribute

Tabulka definující atributy uživatelů. Atributy jsou definovány mimo tabulku User proto, aby se mohly nové atributy definovat podle potřeby jednotlivého nasazení.

Název atributu	Popis

Id	Jednoznačný identifikátor definice atributu
Name	Jméno definice atributu. Musí být unikátní.
Description	Popis definice atributu.

Databázová tabulka 2

- UserAttributeValue

Vazební tabulka definující, jaký uživatel má jaké atributy a hodnoty jednotlivých atributů.

Název atributu	**Popis**
Id	Unikátní identifikátor.
Value	Hodnota atributu.

Databázová tabulka 3

Tabulka UserAttributeValue je vazební tabulkou mezi tabulkami User a UserAtrtibut. Díky tomuto spojení může mít každý uživatel jiné atributy. Lze definovat atributy jako „LoginName“, „LoginName1“ a „loginName2“ a všechny je přiřadit jednomu uživateli. Každý řádek UserAttributeValue má vazbu na jeden řádek tabulky User a jeden řádek tabulky UserAttribute.

- Endpoint

Tabulka připojených koncových systémů. Každý systém, spravovaný aplikací, se zde musí nacházet.

Název atributu	**Popis**
Id	Unikátní identifikátor.
Name	Jméno koncového systému v aplikaci. Musí být unikátní.
Type	Typ koncového systému.
Address	Ip adresa nebo hostname koncového systému.
proxy_user	Název účtu, pomocí kterého se aplikace přihlašuje ke koncovému systému.
Aliases	Náhradní jména koncového systému, pod kterými se může systém v aplikaci nacházet.
private_key	Soukromá část klíče, generovaného aplikací.
public_key	Veřejná část klíče, generovaného aplikací.

last_key_changed	Doba poslední změny klíče endpointu.

Databázová tabulka 4

- EndpointAtribute

Definice Atributů, které se na jednotlivých koncových systémech mohou vyskytovat.

Název atributu	**Popis**
Id	Jednoznačný identifikátor definice atributu.
Name	Jméno definice atributu. Musí být unikátní.
Description	Popis definice atributu.
multi_value	Může atribut nabývat více hodnot?

Databázová tabulka 5

- MirrorAccount

Každému účtu nalezenému na koncovém systému vytvoří aplikace takzvaný zrcadlový účet „mirror account“ podle definovaných atributů. Zrcadlové účty jsou následně připojovány k účtu uživatele.

Název atributu	**Popis**
Id	Jednoznačný identifikátor zrcadlového účtu.
Managealbe	Je účet spravovaný aplikací?
sync_date	Datum poslední synchronizace účtu.
update_date	Datum poslední změny v atributech Zrcadlolého účtu.

Databázová tabulka 6

- AccountAttributeValue

Vlastní hodnota atributu zrcadlového účtu je uložena ve vlastní tabulce. To umožňuje, aby měl každý zrcadlový účet definovanou vlastní sadu atributů podle toho, jaké atributy jsou podporovány koncovým systémem.

Název atributu	**Popis**
Id	Unikátní identifikátor.
Value	Hodnota atributu.

Databázová tabulka 7

- Connector

Přístup aplikace na koncové systémy je zprostředkován pomocí konektorů.

Název atributu	Popis
Id	Unikátní identifikátor.
Name	Jméno konektoru.
Class	Třída podporující povinné API konektorů.

Databázová tabulka 8

- KeyStore

Definuje místo, kam aplikace ukládá klíče pro daný koncový systém.

Název atributu	Popis
Id	Unikátní identifikátor.
Name	Jméno Úložiště.
Description	Popis.

Databázová tabulka 9

- KeyStoreAttribute

Definice atributů, které mohou používat Úložiště klíčů.

Název atributu	Popis
Name	Jméno atributu.
Description	Popis.

Databázová tabulka 10

- KeyStoreAttributeValue

Název atributu	Popis
value	Hodnota atributu

Databázová tabulka 11

4.11.2 Vysvětlení návrhu

4.11.2.1 Koncové systémy

Každý koncový systém bude zanesen do tabulky „Endpoint“. Podle atributu „type“ bude aplikace rozeznávat, o jaký druh koncového systému se jedná a jaký se má tedy použít konektor pro připojení. V tabulce se také nacházejí všechny informace potřebné pro to, aby se aplikace připojila ke koncovému systému. Po ověření spojení s koncovým serverem si aplikace načte všechny účty, které na koncovém systému existují a podle schématu vytvoří ke každému účtu odpovídající zrcadlový účet. Schéma je definováno pomocí vazby tabulky „Endpoint“ na tabulku „AccountAttribute“. Schéma je kontrolováno konektorem, kterým se aplikace ke koncovému účtu připojuje. Každý Zrcadlový účet má atributy podle definice schématu a iniciální hodnoty, které byly nalezeny na koncovém systému. Pomocí atributu „manageable“ bude aplikaci řečeno, jestli smí být Zrcadlový účet připojen k účtu uživatele, nebo ne. Vlastní konektor se bude starat o to, aby účty na koncových systémech odpovídaly jím párovaným Zrcadlovým účtům.

4.11.3 Uživatelé

Každý uživatel aplikace musí být zanesen do tabulky „User“. Uživatel může mít libovolný počet atributu, definovaných tabulkou „UserAttribute“. Pokud bude mít uživatel jeden atribut definovaný vícekrát, bude aplikace považovat atribut jako vícehodnotový a bude se podle toho s ním snažit pracovat. Toto bude umožněno jen u atributů takto definovaných ve schématu služby. Uživatel bude moci mít přidělen jeden nebo více Zrcadlových účtu a to pomocí dvou druhů vazeb:

1. Klasický účet
2. Sdílený účet

4.11.3.1 Klasický účet

Jako Klasický učet smí být každý Zrcadlový účet přidělen jen k jednomu účtu. U takového typu vazby se bude aplikovat mapovací funkce řešená pomocí vazby mezi tabulkami „UserAttributeValue“ a tabulkou „AccuntAttributeValue“. Mapovací funkce bude zajišťovat, že při změnách atributů v tabulce „UserAttributeValue“ dojde i ke změně souvisejících atributů v tabulce „AccountAttributeValue“.

4.11.3.2 Sdílený účet

Zrcadlový účet připojený k uživateli jako Sdílený účet není spravován pomocí mapovací funkce. Aplikace se stará jen o to, aby se všichni uživatelé, kteří mají Zrcadlový účet přiřazený jako Sdílený účet, na tento účet dokázali připojit. Dochází tedy jen k administraci klíčů a ne k nastavování

atributů. Aplikace bude umožňovat nastavování atributů takového účtu, ale to jedině přes editaci Zrcadlového účtu, nebo připojením takového účtu k jednomu uživateli jako Klasický účet.

4.12 Připojení koncových systémů

4.12.1 Nastavení

Pro vynucení autentizace pomocí dvou klíčů je nutné nakonfigurovat SSHD deamona pomocí souboru „sshd_config“. V souboru je nutné nastavit vlastnost „AuthenticationMethods“ na „publickey,publickey“. Tím je zajištěno, že koncový systém bude vyžadovat přihlášení pomocí dvou SSH klíčů. Tato vlastnost je nicméně dostupná až od OpenSSH verze 6.2. SSHD deamon také nekotroluje, jestli bylo použito k přihlášení dvou různých klíčů. Pokud by uživatel použil dvakrát ten samí klíč, SSHD deamon by považoval autentizaci za úspěšně dokončenu.

Tato vlastnost již byla reportována na Mindrot.org, tvůrci OpenSSH deamona. Byl vytvořen Bug s identifikačním číslem 2323, popisující dané chování. Na základě bugu byl vytvořen oficiální patch pro knihovnu OpenSSH verze 6.7p1. Od plánované verze 6.8 se již bude jednat o standardní chování.

Druhou možností je upravit samotný zdrojový kód OpenSSH deamona. Zdrojové kódy jsou volně ke stažení na oficiálních stránkách projektu. Součástí práce je i upravená verze OpenSSH deanoma verze 6.4.p1.

4.12.1.1.1 Úprava OpenSSH deamona

Zde popsané úpravy byly provedeny na knihovně OpenSSH ve verzi 6.4p1, protože se jedná o první verzi knihovny vyšší než 6.2 (od této verze je podporována dvou faktorová autentizace), které je instalována jako základní verze pro již vydané Linux/Unix operační systémy (například CentOS 7). Popis změn slouží pouze jako doporučení.

Pro uložení již použitého klíče byla použita struktura „Key“ definovaná v samotné knihovně. Spolu s klíčem je ukládána informace o tom, jestli se jedná o druhý faktor autentizace. Informace jsou uloženy ve struktuře „Authctx“ definující kontext uživatele v průběhu autentizace. Definice struktury se nachází v souboru „auth.h“:

```
struct Authctxt {
        sig_atomic_t    success;
        int             authenticated; /* authenticated and alarms cancelled */
        int             postponed;     /* authentication needs another step */
        int             valid;         /* user exists and is allowed to login */
```

```
        int             attempt;
        int             failures;
        int             server_caused_failure;
        int             force_pwchange;
        char            *user;          /* username sent by the client */
        char            *service;
        struct passwd   *pw;            /* set if 'valid' */
        char            *style;
        void            *kbdintctxt;
        char            *info;          /* Extra info for next auth_log */
        void            *jpake_ctx;
        Key             *key;           /* slavik patch save used key */
        Int             second_auth_method; /* slavik patch is second auth method */
#ifdef BSD_AUTH
        auth_session_t *as;
#endif
        char            **auth_methods;     /* modified from server config */
        u_int           num_auth_methods;
#ifdef KRB5
        krb5_context    krb5_ctx;
        krb5_ccache     krb5_fwd_ccache;
        krb5_principal  krb5_user;
        char            *krb5_ticket_file;
        char            *krb5_ccname;
#endif
        Buffer          *loginmsg;
        void            *methoddata;
};
```

Inicializaci a uvolnění paměti pro nově vytvořené proměnné se nachází v souboru „sshd.c“. Inicializace byla umístěna jako první věc po vytvoření samotné struktury:

```
/* allocate authentication context */
        authctxt = xcalloc(1, sizeof(*authctxt));

        /* slavik */
        authctxt->second_auth_method = 0;
```

```
        authctxt->key = NULL;
```

Uvolnění paměti bylo přidáno do metody „cleanup_exit“ určené k uvolnění paměti celé struktury:

```
cleanup_exit(int i)
{
        if (the_authctxt) {
        if (the_authctxt->key != NULL)
                key_free(the_authctxt->key);

                do_cleanup(the_authctxt);
                if (use_privsep && privsep_is_preauth && pmonitor->m_pid > 1) {
                        debug("Killing privsep child %d", pmonitor->m_pid);
                        if (kill(pmonitor->m_pid, SIGKILL) != 0 &&
                           errno != ESRCH)
                                error("%s: kill(%d): %s", __func__,
                                   pmonitor->m_pid, strerror(errno));
                }
        }
#ifdef SSH_AUDIT_EVENTS
        /* done after do_cleanup so it can cancel the PAM auth 'thread' */
        if (!use_privsep || mm_is_monitor())
                audit_event(SSH_CONNECTION_ABANDON);
#endif
        _exit(i);
}
```

Informace o tom, že se program nachází v testování druhého faktoru autentizace je ukládána v souboru „auth2.c“:

```
if (authenticated && options.num_auth_methods != 0) {
        if (!auth2_update_methods_lists(authctxt, method, submethod)) {
        authenticated = 0;
        partial = 1;
        /* uprava pro path slavik */
        authctxt->second_auth_method = 1;
        } else {
                authctxt->second_auth_method = 0;
        }
```

```
}
```

Samotné ukládání klíče je řešeno v souboru „auth2-publickey.c“ v metodě „userauth_pubkey“ určené pro autentizaci pomocí klíče:

```
/* Slavik Patch*/
if (authenticated == 1) {
        debug("slavik patch in method saving key");
        if (authctxt->key == NULL) {
                authctxt->key = key_from_blob(pkblob, blen);
                if (authctxt->key == NULL) {
                        error("userauth_pubkey: cannot decode key: %s", pkalg);
                goto done;
                }
        }
}
```

V této metodě je také provedena kontrola, jestli byl klíč již použit:

```
if (authctxt->second_auth_method && key_equal(authctxt->key, key)) {
        debug("slavik patch key already used REFUSE!");
                goto done;
}
```

Patch se nedá instalovat samostatně. Je nutné kompilovat a instalovat celou knihovnu OpenSSH. Popis instalace i s popisem závislostí se nachází v souboru „INSTALL“.

4.12.2 Konektory

Konektorem je nazývána část aplikace zprostředkovávající komunikaci daemona aplikace s připojenými koncovými systémy. Každý konektor musí splňovat API, které je dáno aplikací. Díky API konektorů je aplikaci jedno, s jakým koncovým systémem komunikuje. Konektory aplikaci také sdělují, jaké akce lze na koncovém systému volat a jaké je defaultní schéma podporovaných atributů.

4.12.2.1 API Konektorů

Operace podporované konektory jsou pro názornost rozděleny do následujících tabulek.povinně podporované operace:

Jméno operace	Vstup	Výstup
supportedOperation		Seznam operací, které lze

		pomocí konektoru na koncovém systému volat. Seznam všech operací, které aplikace zná, bude uveden v následující tabulce.
supportedAttributes		Seznam atributů, které je konektor schopen na koncovém systému administrovat.
defaultAttributeSchema		Seznam atributů, které konektor administruje, pokud není nastaveno v aplikaci jinak.

Operace Konektoru 1

Volitelně podporované operace:

Jméno operace	Vstup	Výstup
createUser	Zrcadlový účet	Vytvoří uživatele podle vstupujícího zrcadlového účtu.
deleteUser	Zrcadlový účet	Smaže na koncovém systému uživatele odpovídajícího zrcadlovému účtu.
updateUser	Zrcadlový účet	Nastaví atributy účtu na koncovém systému tak, aby odpovídali zrcadlovému účtu.
listUsers		Zkontroluje existenci zrcadlových účtů a neexistující vytvoří.
setKey	Zrcadlový účet, List<String>	Nastaví soubor s klíči podle vstupu.
getKey	Zrcadlový účet	Vrátí List<String> nebo null

		podle obsahu souboru s klíči.
deleteKey	Zrcadlový účet	Smaže celý soubor s klíči.
listKeys		Vrátí názvy všech klíčů na koncovém systému.
setEndpointKey	List<String>	Nastaví klíč koncového systému.

Operace Konektoru 2

4.13 Komunikace uživatele s aplikací

S aplikací bude možné z uživatelského hlediska pracovat třemi možnými způsoby.

4.13.1 Webové rozhraní

Každý uživatel se bude moci k aplikaci přihlásit pomocí svého prohlížeče. Uživatel si tímto způsobem bude moci získat seznam přidělených koncových systémů, nastavovat některé atributy, vidět stav svého účtu, žádat o okamžité rozeslání klíče nebo synchronizaci účtu na koncové systémy. Uživatel si tímto způsobem nebude moci stáhnou privátní klíč, protože by nemohlo být zajištěno, že jej uživatel bude mít jen v RAM paměti a ne přímo na disku.

4.13.2 SSH rozhraní

Na server se také uživatel bude moct hlásit pomocí protokolu SSH. Po přihlášení bude uživateli zobrazen jednoduchý shell, v kterém si bude uživatel moci vybrat z následujících možností:

- Stáhnout privátní klíč
- Vygenerovat nový privátní klíč
- Nahrát nový veřejný klíč
- Zažádat o synchronizaci na koncové systémy
- Zobrazit stav účtu na daném koncovém systému
- Vypsat aktuální Known-Host

K tomu, aby si mohl uživatel stáhnout privátní klíč, je nutné, aby měl ve chvíli přihlášení k daemonu aplikace zapnutý SSH Agent a „agent forwarding“.

4.13.3 Tlustý klient

Toto je preferovaný způsob jak komunikovat s daemonem aplikace. Daemon aplikace bude mít pro podporu tlustých klientů spuštěnou webovou službu. Služba bude vyžadovat, aby byly zdrojové kódy klienta podepsány certifikátem aplikace. Dále bude muset být klient spojený s SSH

agentem, ve kterém se bude nacházet privátní klíč uživatele. Pomocí klienta bude uživatel schopen vykovávat všechny úkony, které daemon aplikace dovoluje. Uživatel tedy bude moci:

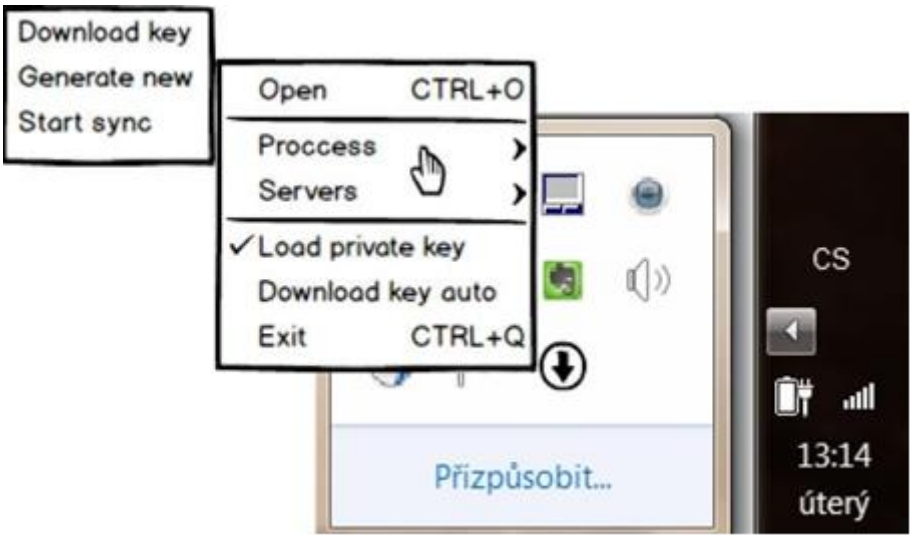
- Nahrát nový veřejný klíč
- Vygenerovat nový klíčový pár
- Stáhnout privátní část klíče do agenta
- Zobrazení stavu účtu v aplikaci
- Zobrazení stavu účtu na přidělených koncových systémech
- Listovat přidělené koncové systémy
- Aktualizovat vlastní soubor s Known-Host
- Zažádat o okamžitou synchronizaci účtu na koncový systém
- Zažádat o okamžité rozeslání klíče na přidělené koncové systémy
- Upozornění na konec session uživatele

Tlustý klient bude schopen (po nastavení) spouštět automaticky SSH klienty, takže si uživatel jen zvolí z nabídky koncových systémů a v tlustém klientovi a ten vytvoří pomocí nastaveného SSH klienta a SSH agenta spojení s daným koncovým systémem.

Implementace Tlustého klienta by měla podporovat běh v rámci takzvaného menu pro systémové ikony nebo oznamovací ikony. Provozováním takovéhoto klienta má mnoho výhod. Takováto architektura umožňuje spuštění jen jedné instance Tlustého klienta najednou. Tlustý klient ale může běžet nepozorovaně na pozadí a uživatel si ji může v případě potřeby vyvolat jednoduše z menu. Tlustý klient také může pravidelně kontrolovat stav serveru a stahovat z něj potřebné informace o změně práv uživatele, změně serverových klíčů nebo o konci časové platnosti staženého klíče.

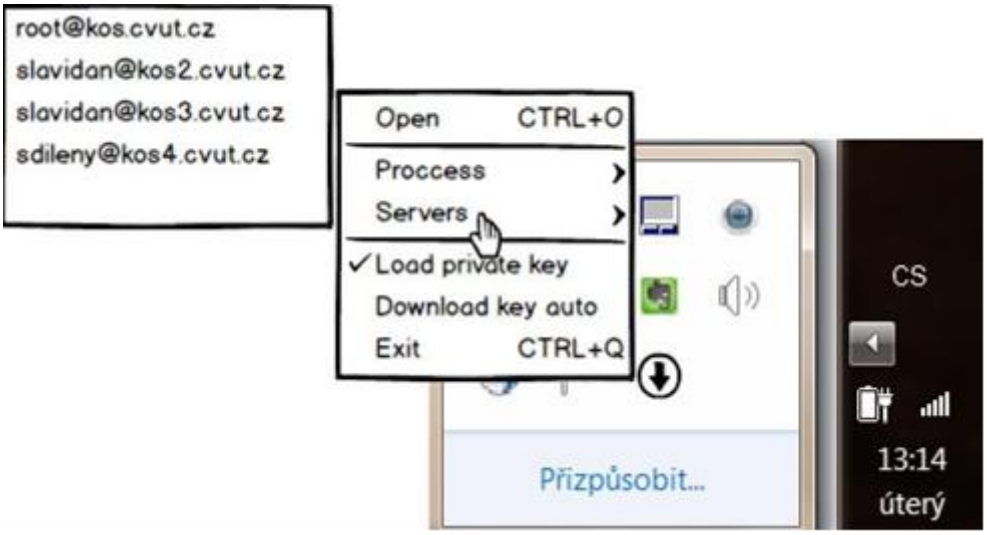
Klient návrh 1

Tlustý klient bude také umožnovat vyvolání nejčastěji používaných procesů jednoduše z menu. Díky tomu bude uživatel ke stažení klíče potřebovat tři kliknutí myši.



Klient návrh 2

Další užitečnou schopností Tlustého agenta bude možnost listovat účty, na které má uživatel přístup. Z menu půjdou přístupy jednoduše vylistovat a po poklepání se předají informace do SSH Klienta a ten vytvoří spojení.



Klient návrh 3

4.14 Komunikace Administrátora s aplikací

Administrace daemona aplikace bude možná třemi způsoby. Každý způsob bude nabízet plnohodnotné administrační nástroje. Administrátor bude moci provádět veškeré úlohy pomocí libovolně zvoleného přístupu.

4.14.1 Úkony spojené s administrací

Možnosti administrátora jsou pro přehlednost uvedeny v následující tabulce.

- Administrace uživatelů

Akce	Popis
createUser	Vytvoří nový učet v aplikaci.
modifyUser	K vybranému účtu se mohou párovat existující definice tributů a nastavovat jejich hodnoty.
deleteUser	Smaže vybraného uživatele z aplikace.
linkAccount	Vytvoří vazbu mezi uživatelem a Zrcadlovým účtem. Při této operaci se volí, jakým typem vazby bude Zrcadlový účet párován (klasickým nebo sdíleným) a vytváří se mapovací funkce mezi tributy uživatele a tributy Zrcadlového účtu. Po provedení operace, pokud je uživatel správně nastaven a jeho účet je platný, je u živateli umožněno přihlásit se na koncový systém.
unlinkAccount	Operace zruší vazbu mezi uživatelem a Zrcadlovým účtem. Po provedení této operace jsou uživatelovi klíče z koncového systému odebrány, aby se uživatel k danému účtu nemohl přihlásit.
searchForSimilar	Spustí proces Párování účtů. Proces je popsán v kapitole Párování účtů.

Možnosti administrátora 1

- Administrace Zrcadlových účtů

Akce	Popis
createAccount	Vytvoří Zrcadlový účet podle schématu daného koncového systému.

modifyAccount	Upraví Zrcadlový účet. Zde bude možnost měnit atributy účtu a připojené definice atributů účtu. Při této operaci je také možné nastavit účet jako neadministrovatelný a tím zabránit aplikaci, aby na něm prováděla jakékoliv úpravy.
deleteAccount	Smaže Zrcadlový účet z aplikace a jemu odpovídající účet z koncového systému.

Možnosti administrátora 2

- Administrace koncových systémů

Akce	Popis
connectEndpoint	Připojí nový koncový systém pomocí zvoleného konektoru.
disconnectEndpoint	Odpojí koncový system.
modifyEndpoint	Změní atributy koncového systému. Zde se upravuje také schéma atributů koncového systému.
synchronizeAccounts	Operace provede update nad všemi Zrcadlovými účty. Touto operací se zajišťuje, že jsou Zrcadlové účty shodné s jejich odpovídajícími účty na koncovém systému.
Reconcile	Provede operaci „Rekonciliace“ popsanou v kapitole Rekonciliace.
changeKey	Změní klíč na koncové službě.
checkKeys	Spustí operaci „Kontrola klíčů“ popsanou v kapitole Kontrola klíčů.

Možnosti administrátora 3

- Administrace atributů uživatele

Akce	Popis
createAttributeDefinition	Vytvoří novou definici atributu.

deleteAttributeDefinition	Smaže vybranou definici atributu.
modifyAttributeDefinition	Změní vybranou definici atributu.

Možnosti administrátora 4

- Administrace konektorů

Akce	**Popis**
loadConnector	Načte nový konektor a provede jeho validaci.
deleteConnector	Smaže vybraný konektor.

Možnosti administrátora 5

4.14.2 Webové rozhraní

Administrátor bude mít možnost se k daemonu aplikace přihlásit pomocí webového prohlížeče. Aplikace bude mít grafické rozhraní pro umožnění všech administračních úkonů.

4.14.3 Příkazový řádek

Po přihlášení na server, kde běží daemon aplikace, bude moci administrátor volat klasické shellové příkazy. Pomocí příkazů a jejich přepínačů bude administrátorovi umožněna kompletní administrace aplikace. Veškeré příkazy i výstupy zde budou striktně textové. Tento způsob administrace není preferovaný, ale aplikace ho bude podporovat pro případy, kdy nebude povoleno přímé internetové připojení k aplikaci, nebo bude nutné provést časově náročnější úlohy, u kterých mohou při webovém přístupu nastat problémy s časovým ukončením spojení.

4.14.4 Webová služba

Daemon aplikace bude mít webovou službu pro umožnění administrace za pomoci jiného produktu. Tam, kde se cíleně bude aplikace vyskytovat, bude také s velkou pravděpodobností nasazen nějaký HR (human resources) nástroj pro správu Identit. Tímto nástrojem může být jak klasické IdM, tak SAP. Pro připojení aplikace k nástroji pro správu identit bude nutné vytvořit speciální konektor. Tento konektor bude využívat daemonem aplikace podporovanou webovou službu.

4.15 Opakované úlohy

V aplikaci se bude dát nastavit pravidelné opakování některých administračních úloh. Nastavitelné úlohy jsou rozděleny podle objektu, nad kterým jsou provedeny.

4.15.1 Nad uživatelem

4.15.1.1 Synchronizace

Proces se dá spouštět jako „plná“ nebo „inkrementální“. Při plné synchronizaci se proces vztahuje na všechny Zrcadlové účty koncového systému. Při inkrementální synchronizaci se proces vztahuje jen na ty účty, které mají datum poslední změny (atribut „update_date“) větší jak datum poslední synchronizace (atribut „sync_date“).

Jedná se o proces, jehož účelem je zajištění, aby všechny spravované Zrcadlové účty odpovídaly svým účtům na koncovém systému. Pomyslně se jedná o směr z aplikace do koncového systému. Daemon aplikace iteruje nad Zrcadlovými účty koncového systému a porovnává je s účty na koncovém systému. Pokud najde rozdíl ve spravovaných atributech, rozdíl zapíše do logu a atribut na koncovém systému napraví. Synchronizace bude podporovat nastavení „reportOnly“, při kterém budou jen reportovat zjištěné změny.

4.15.1.2 Rekonciliace

Rekonciliace je proces, při kterém si daemon aplikace načte všechny účty z koncového systému a hledá k nim odpovídající účty v aplikaci. Proces vytváří vazby mezi účty na koncovém systému a existujícími Zrcadlovými účty. Pomyslně se jedná o směr z koncového systému do aplikace. Daemon aplikace porovná účty a reportuje stav, ve kterém se nacházejí.

Stavy rekonciliace:

Akce	Popis
Confirmed	Byl nalezen účet na koncovém systému i jemu odpovídající Zrcadlová účet v aplikaci.
Deleted	Na koncovém systému nebyl nalezen účet odpovídající zrcadlovému účtu.
Unmatched	Na koncovém systému byl nalezen účet, který neodpovídá žádnému zrcadlovému účtu.

Rekonciliace stavy 1

Nejdůležitějším stavem procesu je stav „Unmatched“. tento stav říká, že na koncovém systému vznikl účet bez zásahu aplikace. Takovéto účty může administrátor vyřešit dvěma způsoby:

1. Nechat účet smazat
2. Vytvořit odpovídající zrcadlový účet v aplikaci.

Při spuštění procesu lze nastavit automatické řešení všech účtu ve stavu „Unmatched“ podle jedné z možností.

4.15.1.3 Párování účtů

Účelem tohoto procesu je usnadnění přiřazování Zrcadlových účtů k uživatelům. Typickým případem je připojení nového koncového systému, na kterém jsou již mnozí uživatelé založeni. Správně nastavený proces pak automaticky přiřadí již existujícím uživatelům v aplikaci Zrcadlové účty nově připojeného koncového systému.

Párování účtů je proces, při kterém aplikace hledá takové Zrcadlové účty, které odpovídají vstupním podmínkám. Jako vstupní podmínky se používá sada vybraných uživatelských definic atributů. Mezi atributy aplikace vkládá podmínku OR. Ke skupině atributů uživatele se vybírá jeden atribut koncového systému. Příklad:

V aplikaci existují nadefinované atributy „login1“ a „login2“. Uživatel „Daniel.Slavík“ má obě definice přiřazeny a nastaveny na hodnoty „daniel.slavik“ a „slavidan“. Vybraný atribut koncového systému je „loginName“. Aplikace tedy vybere všechny Zrcadlové účty, které mají atribut „loginName“ o hodnotě buď „daniel.slavik“ nebo „slavidan“.

Párování uživatelů reportuje uživatele v těchto stavech:

Stav	Popis
Unassigned	Aplikace našla účet, který odpovídá zadaným podmínkám a uživatel jej nemá přiřazen.
Duplicated	Učet vyhovuje více uživatelům.
Assigned	Účet vyhovuje zadání, ale je již přiřazen jako klasický účet jinému uživateli.

Synchronizace stavy 1

Administrátor může přiřadit Zrcadlové účty ve stavu „Unassigned“ uživateli jako klasický účet, nebo sdílený. Zrcadlové účty ve stavu „Assigned“ se dají přiřadit jen jako sdílené účty. U obou stavů lze proces nastavit tak, aby přiřazení dělal automaticky. Účty ve stavu „Duplicated“ je možně přiřadit jedině zásahem administrátora.

4.15.1.4 Rotace klíčů

V daemonu aplikace půjde nastavit takzvaná „session uživatele“. Pod tímto pojmem je schována doba, po kterou bude moci uživatel využívat klíč generovaný aplikací. Při stažení klíče daemon aplikace nastaví, pokud již nastavený není, atribut uživatele „last_download_key“ na aktuální

datum a čas a vygeneruje náhradní klíč, který si uloží. Proces Rotace klíčů kontroluje dobu, která uplynula od časové značky v atributu uživatele a pokud čas přesáhl nastavenou dobu, odebere ze všech koncových systémů uživatele veřejnou část klíče generovaného daemonem aplikace a rozešle část novou. Pokud se chce uživatel na koncové systémy přihlásit, musí si stáhnout klíč nový. Tím bude zajištěno, že se uživatel bude muset alespoň jedenkrát po zvolenou dobu přihlásit k daemonu aplikace.

4.15.2 Nad koncovými systémy

4.15.2.1 Kontrola klíčů

Daemon aplikace se bude pravidelně v nastavených intervalech připojovat ke všem koncovým systémům a kontrolovat klíče všech účtů. Pokud daemon nalezne klíč, který nezná, situaci zaloguje a klíč odebere. Úloha si nebude všímat neznámých klíčů u účtů, jejichž odpovídající Zrcadlové účty mají nastaven atribut „manageable“ na „false“. Pokud daemon na koncovém systému nenalezne klíč uživatele, který by tam měl být, situaci zaloguje a klíč nahraje. Proces bude podporovat spuštění v „readOnly“ módu, při kterém bude jen reportovat zjištěné nedostatky.

4.15.2.2 Výměna klíčů server

Podobným způsobem jako daemon aplikace spravuje klíče uživatelů, bude spravovat klíče připojených koncových systémů. Podle atributu „last_key_changed“, ve kterém se bude nacházet datum a čas poslední změny, vyhodnotí aplikace uplynulou dobu. Pokud uplynulý čas přesáhne dobu nastavenou v proměnné „EndpointKeyExpiration“, daemon aplikace vygeneruje nový klíč a nastaví jej na koncový systém.

4.15.2.3 Known-Hosts

Protože bude daemon aplikace schopný měnit klíče připojených koncových systému, bude docházet k tomu, že všichni uživatelé, kteří se na koncový systém přihlásili předtím, než byl klíč změněn, budou na změnu upozorněni. To je nežádoucí jev. Upozornění na změnu pro uživatele znamená, že se přihlašuje na jiný server, než na který se přihlašoval předtím. Tímto způsobem se indikuje útok „Man In The Middle“. Proto, aby nevznikaly poplašné zprávy, bude aplikace přes tlustého klienta schopna editace souboru „known-host“ a uživatelově pracovní stanici. Tlustý klient si při přihlášení na server zkontroluje, jestli má soubor aktuální, a pokud ne, soubor aktualizuje. Tlustý klient nebude přehrávat celý soubor (došlo by k smazání záznamů ze systémů, které nejsou pod správou aplikace), ale jen přepisovat aktualizované řádky. Klíče se budou uchovávat v databázi u koncových systémů a aktualizace se bude vypočítávat z časových značek posledních změn klíčů na koncových systémech.

4.15.2.4 Výměna vlastního klíče daemona aplikace

Pro zabránění odcizení klíče, kterým se daemon aplikace přihlašuje na koncové systémy, se bude klíč v pravidelných intervalech měnit. Daemon aplikace si vygeneruje nový klíč, nahraje jeho veřejnou část na server, vyzkouší, zda je schopný se za pomoci nového klíče na koncový systém přihlásit a pokud se všechny předešlé kroky podaří, odstraní klíč starý. Délka platnosti klíče daemona aplikace se bude dát nastavit pomocí property „AplicationKeySession“.

4.16 Logování

Daemon aplikace bude vytvářet několik na sobě nezávislých logů. Jejich význam je popsán níže.

4.16.1 AuditLog

Do auditlogu spadají všechny události provedené nad uživateli. Bude zde zaznamenán celý životní cyklus uživatele aplikace. Budou zde logovány následující události:

- Přihlášení k daemonu aplikace
- Vytvoření účtu
- Změna účtu
- Smazání účtu
- Změna klíče
- Změna vazby mezi účtem a Zrcadlovým účtem
- Rozeslání klíče na koncové systémy
- Smazání klíče z koncových systémů

4.16.2 SysLog

SysLog slouží pro administrátory aplikace. Budou zde zaznamenány všechny interní zprávy a chyby aplikace.

4.16.3 AccountProcessLog

Při běhu opakovaných úloh koncového systému jako je Rekonciliace, Synchronizace, Párování uživatelů apod. bude vytvářen speciální log jménem „AccountProcessLog“. Z tohoto logu budou dohledatelná všechna automatická párování účtů, automatická vytvoření Zrcadlových účtů, reportované změny mezi atributy Zrcadlového účtu a účtu na koncovém systému i všechny účty založené bez vědomí aplikace.

4.16.4 KeyCheckLog

Proces kontroly klíčů bude natolik podstatný, že jeho výsledek bude zaznamenán do vlastního logu. V logu bude dohledatelné, jaké neznámé klíče byly na koncových systémech nalezeny. Při úpravách souborů s klíčem bude log obsahovat původní změněný soubor pro případ, kdy automatické smazání klíčů bude nežádoucí. Tento log je určen jen pro administrátory aplikace, ale bude z něj možné exportovat informace o neznámých klíčích pro použití v jiných aplikacích.

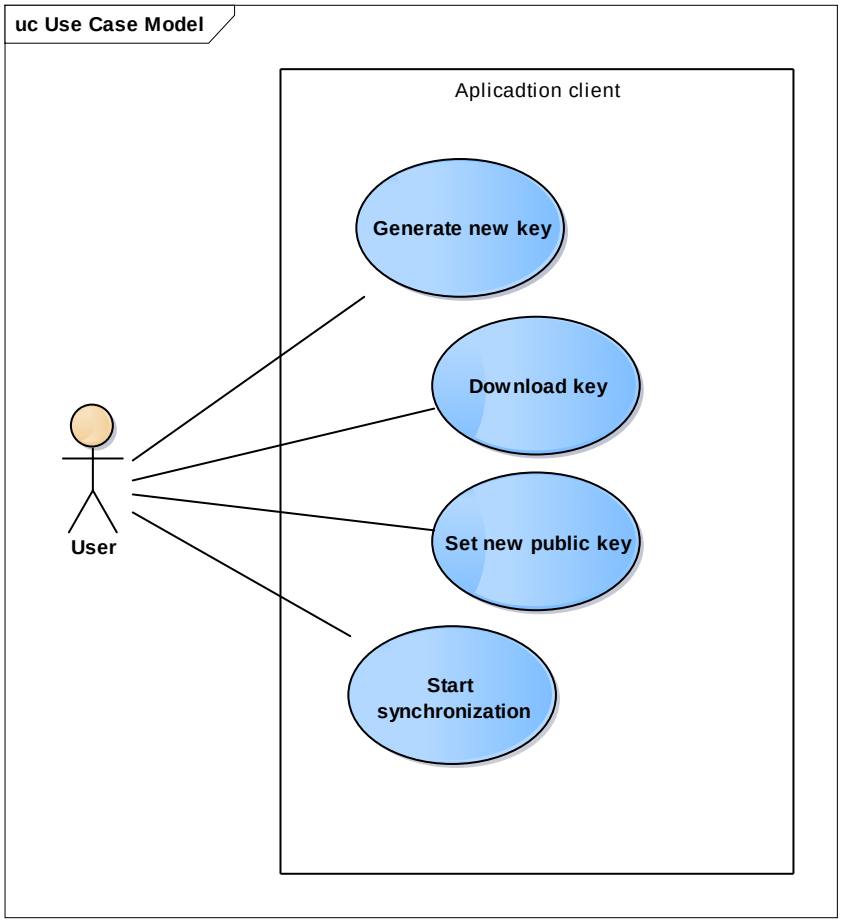
4.17 Připojené nespravované systémy

Existují koncové systémy, u kterých nechceme, aby je aplikace spravovala, přesto ale chceme, aby uživatelé pod klíčem generovaným daemonem aplikace na ně přístup měli. Takovýmto systémům se se říká „Připojené nespravované systémy“. Takovýto systém se bude z hlediska aplikace lišit jen konektorem, který jej připojuje. Konektor nebude přímo spravovat účty, ale jen informovat administrátora koncového systému o úkonech, které mají být vykonány. Při všech opakovaných procesech bude konektor vracet informace z databáze daemona aplikace, takže každý proces skončí bez žádné chyby. Jediným problémem u takto připojených systému je, že pokud bude uživatelská session nastavena moc krátce, bude pro administrátora koncového systému velice nepohodlné takový systém spravovat (například při nastavení expirace uživatelovy session na jeden den by znamenalo, že administrátor bude muset každý den nahrávat na servery nové klíče.).

5 Implementace

Protože je návrh aplikace příliš velký pro implementaci jedním člověkem, naimplementoval jsem jen jeho podmnožinu. Samotný kód není začátkem plnohodnotné implementace, ale jedná se o takzvaný „Proof of concept“, ve kterém se snažím dokázat, že základní principy, na kterých je aplikace založena, jsou za použití prostředků dnešní doby implementovatelné a to bez nutnosti úprav a snížení bezpečnosti navrhované aplikace.

5.1 Popis případu užití



Případ užití 1

Hlavním úkolem aplikace je umožnit přístup všem svým uživatelům na připojené systémy a to tak, aby museli zadávat přihlašovací údaje jen jedinkrát. Základní cyklus užívání aplikace tedy je, že si uživatel nechá vygenerovat klíč, ten si stáhne do SSH Agenta a pomocí něho se přihlásí na všechny spravované systémy. Implementovaná část aplikace se tedy skládá ze tří částí: Připojovaný systém, daemon aplikace a tlustý klient. Implementovaná část aplikace umožňuje uživateli čtyři základní případy užití:

1. Generování nového klíče.

Uživatel může zaslat daemonu aplikace požadavek na vygenerování nového privátního i veřejného klíče.

2. Stažení klíče.

Pokud má uživatel již vygenerován klíč, může si jej nechat poslat do svého SSH Agenta.

3. Nahrání nového veřejného klíče.

Uživatel si bude moci nahrát veřejnou část vlastního klíče do daemona aplikace. Tato část klíče se stane součástí celkového veřejného klíče, který bude nahráván na připojené systémy.

4. Spuštění procesu synchronizace

Implementována je jen malá část procesu a to ta, při které dochází k rozeslání celého veřejného klíče na připojené systémy.

5.1.1 Připojený system

Jako vzorový systém pro připojení do implementované částí aplikace jsem si vybral Ubuntu verze 13.10. Server má instalovánu knihovnu OpenSSH verze 6.2p2.

5.1.1.1 Konfigurace systému

Aby mohl být systém připojen k aplikaci, je nutné jej k tomuto nakonfigurovat. Na serveru jsem založil uživatele, kterým se bude na server přihlašovat daemon aplikace (uživatel se jmenuje „apProxy“). Soubor „/etc/ssh/sshd_config“ byl upraven tak, aby vyhovoval potřebám daemona. Provedené úpravy s jejich popisem a zdůvodněním se nacházejí níže.

1. AuthorizedKeysFile /etc/ssh/authorized_key/%u-key

Nastavení serveru tak, aby vyhledával klíče přihlašovaných uživatelů ve složce „/etc/ssh/authorized_key“. Samotné soubory s veřejnými klíči jsou pro každého uživatele unikátní a skládají se ze jména uživatele (pomocí %u) a řetězce „-key“. Například úplná cesta ke klíči proxy uživatele aplikace je „/etc/ssh/authorized_key/apProxy-key“.

2. AuthenticationMethods publickey,publickey

Tímto je server nastaven tak, aby k přihlášení uživatele vyžadoval dvě úspěšné autentizace pomocí SSH klíče.

3. Match User apProxy AuthenticationMethodspublickey

Protože je zbytečné, aby se daemon aplikace přihlašoval na server pomocí dvou klíčů (nepřináší to žádnou bezpečnostní výhodu) je proxy uživatel z této povinnosti vyjmut.

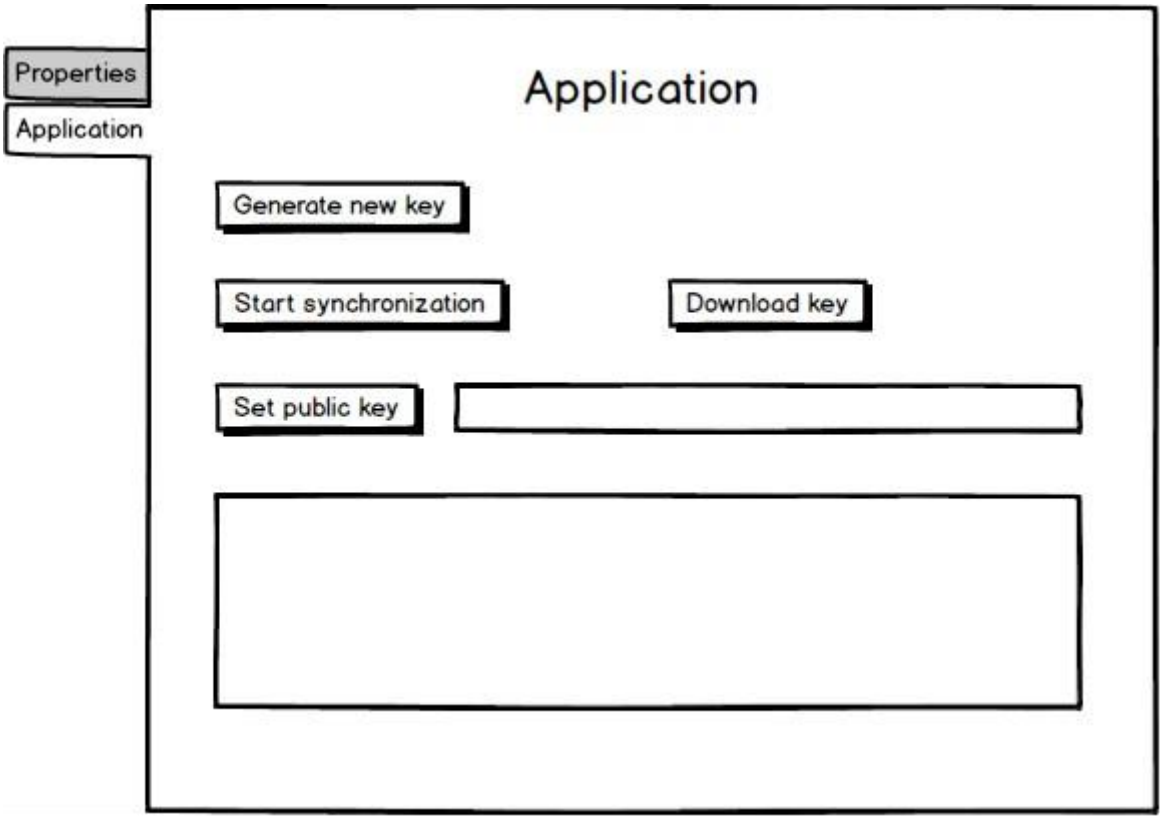
K tomu, aby všichni uživatelé směli používat klíče v jedné složce, je nutné, aby do ní měli všichni přístup a mohli klíče listovat. Samotné klíče mají práva nastavena tak, aby je směl editovat jedině vlastník. Ostatnímu světu jsou otevřeny jedině pro čtení. Vlastníkem všech klíčů je „root“.

5.1.2 Daemon aplikace

Pro server s aplikací jsem zvolil Debian verze 7.4 (wheezy). Pro pohodlnější ovládání má server nainstalováno grafické rozhraní. Na serveru je nainstalován aplikační server Glassfish verze 4. Na tomto aplikačním serveru je daemon aplikace vystavena. Pro potřebu změny mezi formáty klíčů je na serveru nainstalován balík „puttygen-tool“.

5.1.3 Tlustý klient

Pro implementaci tlustého klienta jsem zvolil programovací jazyk JavaFX. Jedná se o jednoduchou aplikaci zprostředkovávající komunikaci s webovou službou. Na rozdíl od původního návrhu klienta, kde měl klient běžet v oznamovací oblasti, je klient vytvořen jako desktopová aplikace. To jsem udělal z důvodu jednoduššího testování.



Návrh klienta 1

Tlustý klient se skládá ze dvou obrazovek. První slouží ke komunikaci s webovou službou. Druhá obrazovka umožňuje nastavení klienta. Pro chod klienta je nutné sdělit nastavit potřebné informace:

1. Application address

Ip adresa serveru, na kterém běží daemon aplikace.

2. Port

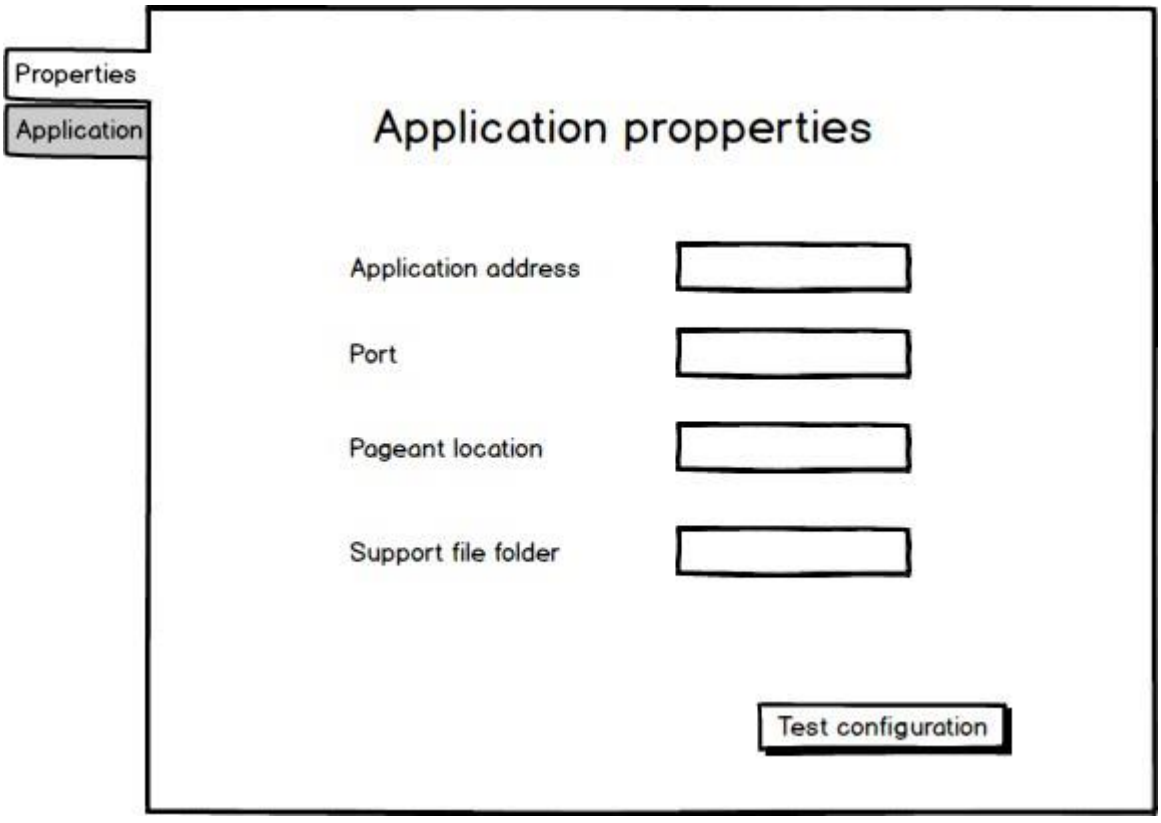
Port, na kterém daemon aplikace naslouchá.

3. Pageant location

Úplná cesta k souboru pageant.exe na souborovém systému uživatele. Tlustý klient bude přímo do agenta nahrávat klíče, které mu budou daemonem zaslány.

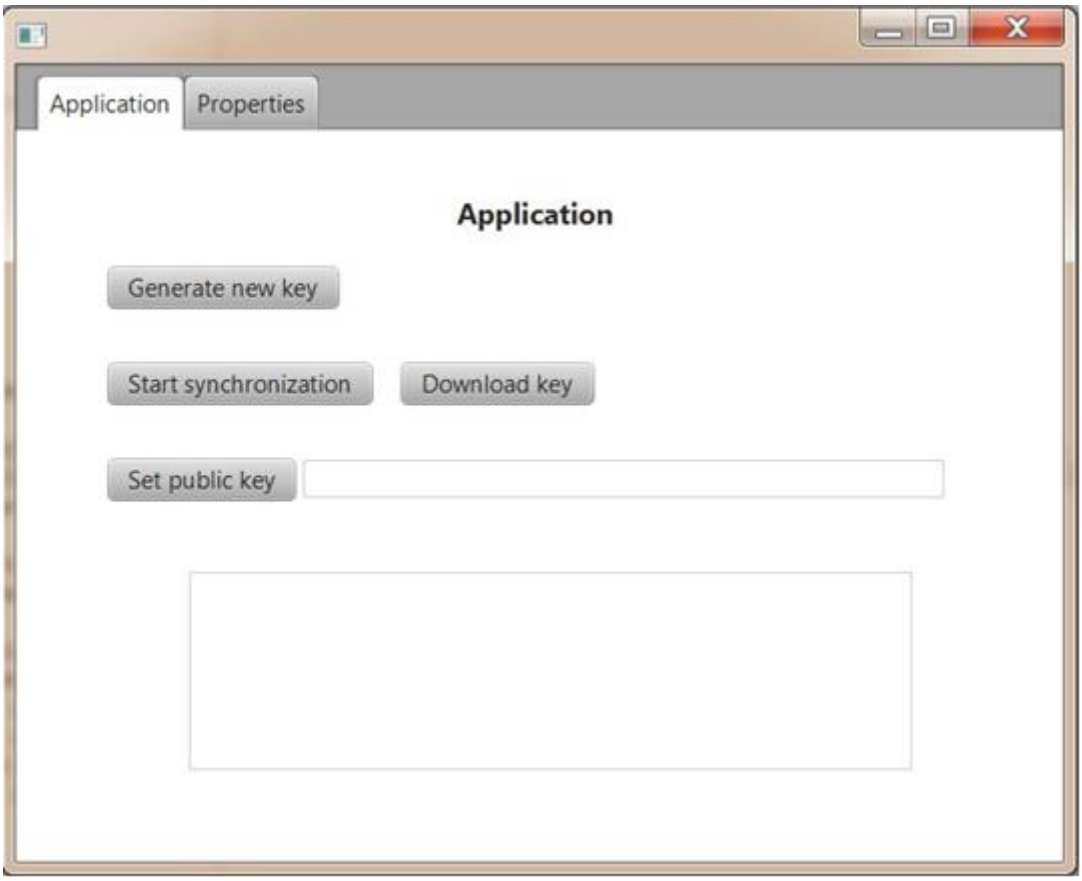
4. Support file folder

Protože Pageant běžící na operačním systému Windows podporuje jen jediný příkaz z příkazové řádky a tím je nahrání klíče ze souboru, musí aplikace vytvořit soubor, do kterého klíč nahraje. Tento soubor po nahrání okamžitě odebere (před odebráním je soubor náhodně změněn, aby nemohlo dojít k jeho obnovení).



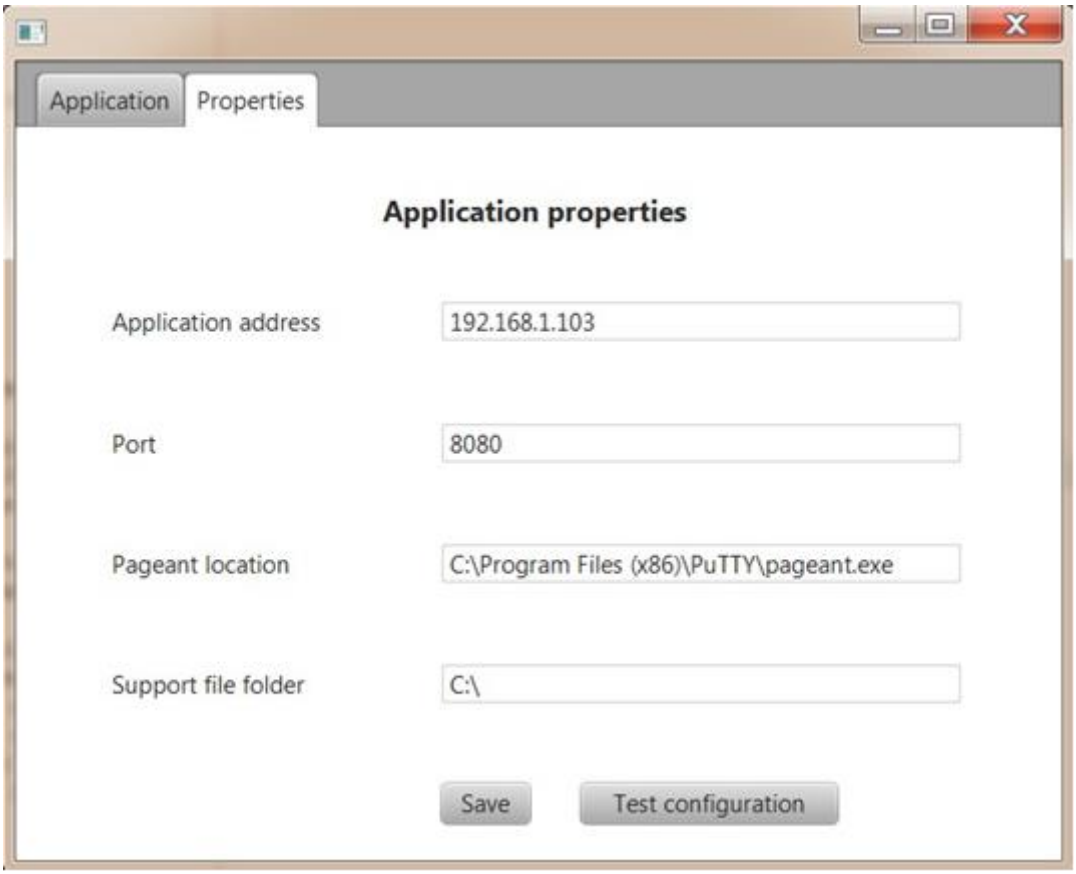
Návrh klienta 2

Při vlastní implementaci jsem se od návrhu trochu odchýlil. To bylo způsobeno vyjadřovacími schopnostmi jazyka JavaFX. Tlustý klient byl implementován pro spuštění na operačním systému Windows s instalovanou Javou verze 1.7 nebo vyšší. Dále je pro chod nutné mít instalovaný SSH agent Pageant a s ním SSH klienta PuTTY. Obě instalace, jak klienta, tak agenta, se musejí nacházet v jedné složce (pro umožnění jejich vzájemné komunikace mezi sebou).



Realizace klienta 1

Tlustý klient si pro usnadnění používání vytváří soubor s názvem appData.txt, do kterého si ukládá nastavená data pro další spuštění. Všechna tlačítka v první obrazovce jsou zneplatněna (mají nastaven příznak „disable“) dokud uživatel nevyplní požadované informace v záložce Properties.



Realizace klienta 2

5.1.4 Práce s daemonem aplikace

Pro usnadnění testování daemona aplikace jsem vytvořil dva virtuální servery (oba typu VMware), jeden nastaven jako připojovaný systém a druhý s běžícím daemonem aplikace.

Pro otestování funkčnosti návrhu je nutné postupovat v následujících krocích:

1. Spustit virtuální server Ubuntu.

Na server se dá přihlásit přímo pomocí uživatele „dan“ a hesla „rootroot“. Uživatel má přes sudo nastavena všechna práva.

2. Pomocí příkazu „ifconfig“ zjistit ip adresu, která byla serveru Ubuntu přidělena.
3. Spustit server Debian.

Na server se dá přihlásit přímo pomocí uživatele „dan“ a hesla „rootroot“. Uživatel má přes sudo nastavena všechna práva.

4. Pokud není daemon aplikace spuštěn, provést jeho redeploy.
    a. Pomocí prohlížeče se přihlásit na adresu „localhost:4848“.
    b. Přihlásit se do aplikačního serveru pomocí uživatele „admin“ a hesla „adminadmin“.

c. V záložce aplikace stisknout tlačítko „Redeploy“ u aplikace jménem „diplomovaPrace“.
d. Přistoupit na adresu „localhost:8080/diplomovaPrace“ na serveru Debian.
e. Do pole zadat ip adresu serveru Ubuntu a stisknout tlačítko odeslat. Tím dojde k naplnění databáze testovacími daty a připojení serveru Ubuntu, poslouchajícího na zadané adrese.

5. Spustit klienta pomocí příkazu „java –jar clientDip.jar“. Protože klient vyžaduje oprávnění pro zápis některých souborů, je nutné jej spustit s oprávněním administrátora.
6. Do záložky Properties vyplnit adresu serveru Debian, port 8080 a potřebné údaje podle vlastního počítače.
7. Stisknutím tlačítka „Generate new key“ vygenerovat nový klíč.
8. Stisknutím tlačítka „Download key“ stáhnout klíč do agenta.
9. Vygenerovat vlastní klíč.
    a. Asi nejrychlejší způsob je pomocí programu puttygen.
10. Nahrát vlastní klíč do klienta.
11. Zadat cestu k souboru s veřejnou částí klíče do pole a stisknout tlačítko „Set public key“.
12. Stisknout tlačítko „Start synchronization“.

Po těchto úkonech bude mít uživatel v SSH agentovi dva soukromé klíče a jejich veřejné části se budou nacházet na serveru Ubuntu. Klíče se na server nahrávají pro uživatele „dipTest“, pod kterým je nyní možně se přihlásit.

5.2 Neimplementované schopnosti aplikace

Cílem vytvořeného programu bylo prokázání funkčnosti principu přihlašování pomocí dvou klíčů. Z toho důvodu nebyly implementovány některé navržené funkce aplikace. Pokud by se měla aplikace dodělat do prodejného stavu, bylo by nutné doplnit následující funkce.

5.2.1 Konektor

Konektor, vytvořený pro účely testování, podporuje jen metody pracující s klíči. Nebyly implementovány ani metody pro spravování uživatelů, ani metody pro spravování oprávnění. Pokud by měla aplikace skutečně spravovat uživatelské účty a ne jenom rozesílat klíče, bylo by nutné toto doplnit. Bylo by také potřeba vytvořit speciální konektory pro ostatní typy systémů (vytvořený konektor spravuje jen systémy typu Linux a i u těch se může syntax jednotlivých příkazů lišit) které se mají do aplikace připojit (HP, SUSE, AIX, Cisco atd.). Pro usnadnění práce s tvorbou konektorů byla vytvořena šablona zprostředkovávající protokol SSH a connection pool. Stačí tedy, aby konektory byly dědily od třídy „SSHConnectorAbs“ a ne od třídy „ConnectorAbs“.

5.2.2 Tlustý klient

Původní návrh tlustého klienta počítal s tím, že bude spouštěn v takzvaném „systém tray“ módu. To znamená, že půjde spustit jen jedna instance aplikace najednou a ta se bude vyskytovat systémové části pro upozornění. Díky tomu jej bude mít uživatel snadno stále po ruce. Dále by měl tlustý klient v sobě zahrnovat jak SSH Agenta, tak SSH klienta. K jejich instalaci by mělo dojít v průběhu instalace Tlustého klienta. Klient by měl také na pozadí vyvolávat jak SSH Agenta (dělá i nyní), tak SSH Klienta. Klient by tedy měl být schopen uchovávat informace o tom, kam má uživatel přístup v rámci aplikace a uživateli tento seznam přívětivým způsobem zprostředkovat.

K tomu, aby mohli aplikaci užívat i uživatelé jiných operačních systémů jak Windows 7, je potřeba pro tyto systémy Tlustého klienta vytvořit. Klienti využívají webovou službu a tím je jejich implementace nezávislá na programovacím jazyce.

Dále by klienti měli podporovat více druhů SSH Agentů a SSH Klientů (momentálně implementace podporuje jenom PuTTY).

Součástí tlustého klienta má podle návrhu být i autentizace vůči daemonu aplikace. Implementovaná část se přihlašuje pod jedním uživatelem a heslo není kontrolováno. Aby implementace odpovídala návrhu, musel by se klient přihlašovat jménem a heslem, komunikace by musela využívat protokol SSL a uživatel by pro stažení musel zadat passphrase ke stahovanému klíči. Ani jeden ze zabezpečovacích mechanizmů nebyl v rámci testovacího vývoje implementován.

5.2.3 Webová služba pro uživatele

Implementovaná část webové služby dovoluje uživateli jen úkony potřebné k úspěšnému přihlášení. V návrhu je ale popsáno mnohem více. Na implementaci popsaných metod z návrhu závisí schopnosti Tlustého klienta.

5.2.4 Webová služba pro administrátory

Administrátorský přístup nebyl implementován vůbec. Tato webová služba by měla sloužit pro umožnění připojení aplikace na existující řešení Identity manageru v každé instituci. Bez jejího vytvoření nebude možné daemona aplikace ovládat.

5.2.5 Webové stránky

Návrh aplikace počítá s možností, že jej bude administrátor ovládat pomocí webových stránek. Momentální implementace nemá žádné webové stránky, ale je upravena tak, aby se do ní daly jednoduše dodělat. Jako Framework je použit JSF2.0. Implementací webových stránek se značně usnadní a ulehčí administrace aplikace.

6 Závěr

Z úvodní studie, srovnávající již existující produkty, vyplynulo, že navrhovat aplikaci pro správu účtů a autentizaci na Linux/Unix systémy má smysl. Dále studie specifikovala několik funkčních i nefunkčních požadavků, které aplikace musí splňovat, aby byla konkurenceschopná. Aplikace by například neměla používat heslo jako způsob autentizace na koncové systémy a uživatelům musí umožnovat SSO.

Při návrhu aplikace bylo nutné vyřešit několik klíčových prvků architektury jako samotnou autentizaci ke koncovým systémům, způsob ukládání a distribuce klíčů, možnost připojení nestandardních nebo ještě neexistujících Linux/Unix systémů i samotný způsob komunikace uživatele s aplikací. Pro způsob autentizace byla zvolena metoda dvou SSH klíčů, která zajišťuje jak prokazatelnost identity uživatele, tak nepopiratelnost uživatelova přihlášení. Přes tyto výhody zůstává zachována uživatelská přívětivost používání klíčů. Pro umožnění tohoto typu autentizace bylo nutné upravit samotnou knihovnu OpenSSH. O smysluplnosti navrženého řešení mluví i fakt, že úpravy budou začleněny tvůrci knihovny OpenSSH, do knihovny samotné. V závěru byla aplikace implementována ve stádiu Proof Of Concept, ověřující navržený koncept autentizace.

Navržená aplikace sice neumožňuje detailní sledování činnosti uživatele na koncových systémech nebo vyhodnocování jeho chování, ale umožňuje kontrolu nad existencí a platností účtů. Dále řeší samotnou autentizaci uživatelů ke koncovým systémům způsobem, který ještě nebyl nikde použit. Zjednodušeně řečeno se aplikace nestará o uživatele, který prošel pomyslnými dveřmi do systému (autentizací), ale stará se o to, aby na koncových systémech nevznikaly dveře nové, původní byly stále zabezpečeny a nebyly otevřené zbytečně dlouho.

7 Zkratky

ANSI American National Standard Institute. Americký institut zabývající se standardy.

DES Data Encryption Standard. Symetrická šifra.

DSA Algoritmus asymetrického šifrování.

FIPS Federal Information Processing Standards. Agentura zabývající se standardy pro vládu Spojených států.

GID Group Id. Jednoznačný identifikátor skupiny na unixových systémech.

H-D Algoritmus pro výměnu klíčů symetrické šifry, vymyšlený Martinem Hellmanem a Whitfield Diffie.

HTTP Hypertext Transfer Protokol. Protokol využívající se při internetové komunikaci.

IFC Identity Connector Framework. Šablona pro vytváření konektorů do Identity managerů, spravovaná firmou ORACLE.

ISO International Organization for Standardization. Organizace spravující standardy všeho druhu.

JSF Java server faces. Framework pro tvorbu webových stránek.

KDC Key Distribution Center. Server distribuující tickety v protokolu Kerveros

NSA National Security Agency. Bezpečnostní agentura Spojených států

PKCS Publick Key Cryptographic Standard. Skupina standardů pro asymetrickou kryptografii spravovaná firmou RSA Data Security.

PKI Public Key Infrastructure. Infrastruktura pro správu a distribuci asymetrických klíčů.

PKC Publick key infrastructure. Infrastruktura pro správu klíčů asymetrického šifrování.

ROOT Uživatel se všemi právy na unixových systémech.

RSA Algoritmus asymetrického šifrování.

SOAP Simple Object Access Protokol. Protokol pro komunikaci přes http pomocí výměny XML zpráv.

SSH Zabezpečený protokol pro komunikaci s unixovými systémy.

SSH Agent Aplikace pro uchování klíčů v RAM paměti uživatelovy pracovní stanice.

SSH Klient Aplikace pro zprostředkování komunikace s unixovým systémem pomocí protokolu SSH.

SSO Single Sing On. Vlastnost architektur umožňující uživateli po jediném přihlášení využívat více koncových systémů

UID User Id. Jednoznačné identifikátor uživatele na unixových systémech.

XML Simple Object Access Protokol. Protokol pro komunikaci přes http pomocí výměny XML zpráv.

LDAP (Lightweight Directory Access Protocol) Definovaný protokol pro ukládání a přístup k datům na adresářovém server.

RFC (Request For Enhancment) Označení řady standardů a dalších dokumentů popisujících Internetové protokoly, systémy apod.

8 Literatura

[1] FIPS Publication 46 - 3 [online]. 2005 [cit. 2014-05-09]. Dostupné z: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[2] The History of Non-Secret Encription [online]. 2000 [cit. 2014-05-10]. Dostupné z: http://cryptocellar.web.cern.ch/cryptocellar/cesg/ellis.pdf">http://cryptocellar.web.cern.ch/cryptocellar/cesg/ellis.pdf

[3] HELLAMN, Martin a Whitfield DIFFIE. Multiuser cryptographic techniques. Proceedings of the National Computer Conference, 1976.

[4] Oasis PKI. PKI Technical Standards [online]. 30 March 2014 at 11:38 [cit. 2014-04-03]. Dostupné z: http://www.oasis-pki.org/resources/techstandards/

[5] R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, Vol. 21 (2), pp.120–126. 1978. Previously released as an MIT „Technical Memo“ in April 1977. Initial publication of the RSA scheme.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 31.7: The RSA public-key cryptosystem, pp.881–887.

[7] Funding Universe. RSA Security Inc. History [online]. 30 March 2014 at 11:38 [cit. 2014-04-03]. Dostupné z: http://www.fundinguniverse.com/company-histories/rsa-security-inc- history/

[8] History of the Internet. Wikipedia, the free encyclopedia [online]. 4 April 2014 at 20:59 [cit. 2014-04-03]. Dostupné z: http://en.wikipedia.org/wiki/History_of_the_Internet#Historiography

[9] An Overview of Cryptography. Gary C. Kessler [online]. 9 March 2014 [cit. 2014-04-03]. Dostupné z: http://www.garykessler.net/library/crypto.html

[10] Free Encyclopedia of Ecommerce. RSA Data Security [online]. 9 March 2014 [cit. 2014-04-03]. Dostupné z: http://ecommerce.hostip.info/pages/914/RSA-Data-Security.html

[11] IBM. The History of Notes and Domino [online]. 14 November 2007 [cit. 2014-04-03]. Dostupné z: http://www.ibm.com/developerworks/lotus/library/ls-NDHistory/

[12] The Digital Distributed System Security Architecture Morrie Gasser, Andy Goldstein, Charlie Kaufman, Butler Lampson Digital Equipment Corp. 85 Swanson Rd., Boxborough, Mass. 01719 Proc. 12th National Computer Security Conf., NIST/NCSC, Baltimore, 1989, pp 305-319.

[13] Microsoft Research. The Digital Distributed System Security Architecture [online]. January 1989 [cit. 2014-04-03]. Dostupné z: http://research.microsoft.com/apps/pubs/default.aspx?id=68218

[14] EMC. Public-Key Cryptography Standards (PKCS) [online]. 2014 [cit. 2014-04-03]. Dostupné z: http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/public-key-cryptography-standards.htm

[15] FIPS. FIPS 186-3. Gaithersburg: nformation Technology Laboratory National Institute of Standards and Technology, 2009. Dostupné z: http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[16] Pokročilá kryptologie: Kryptografie eliptických křivkek. Edux.fit.cvut.cz [online]. 2011 [cit. 2014-04-03]. Dostupné z: https://edux.fit.cvut.cz/oppa/MI- KRY/prednasky/prednaska11_12.pdf

[17] HEWITT, Paul. A brief history of elliptic curves. December 5, 2005. Dostupné z: http://livetoad.org/Courses/Documents/132d/Notes/history_of_elliptic_curves.pdf

[18] LITTLE HEATH, Thomas. A History of Greek Mathematics. Clarendon Press, 1921. Dostupné z: https://archive.org/details/ahistorygreekma00heatgoog

[19] National institutes of standards and technology. Digital Signatures [online]. 2006 [cit. 2014-04-05]. Dostupné z: http://csrc.nist.gov/groups/ST/toolkit/digital_signatures.html

[20] Application of X.500 and X.700 Standards for Supporting Personal Communications in Distributed Computing Environments. Distributed Computing Systems, 1995., Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of, 1995.

[21] Kenneth Geisshirt: Pluggable Authentication Modules: The Definitive Guide to PAM for Linux SysAdmins and C Developers, Packt Publishing, ISBN: 978-1-84719-022-2, 2007

[22] Symeon (Simos) Xenitellis: The Open-source PKI Book A guide to PKIs and Open-source Implementations, http://ospkibook.sourceforge.net/docs/OSPKI-2.4.6/OSPKI.pdf , 2000

[23] Brett McLaughlin: Java & XML Data Binding, O'Reilly Media, ISBN: 978-0-596-00278-7, 2002

[24] NetIQ: NetIQ Privileged User Manager. NetIQ [online]. 2014 [cit. 2014-05-06]. Dostupné z: https://www.netiq.com/products/privileged-user-manager/features/secure-auditable.html

[25] CyberArk. CyberArk [online]. 2012 [cit. 2012-08-02]. Dostupné z: http://www.cyberark.com/product-detail/pim-on-demand-privileges-manager-unix-linux

[26] Protecting Access to Sensitive Resources with Oracle Privileged Account Manager. Oracle Privileged Account Manager [online]. 2013 [cit. 2014-05-06]. Dostupné z: http://www.oracle.com/technetwork/middleware/id-mgmt/overview/opam-wp-11gr2-1697093.pdf

[27] SELinux. Centos [online]. 2014 [cit. 2014-05-06]. Dostupné z: http://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-selinux.html

[28] RedHat: Customer portal [online]. 2014 [cit. 2014-05-09]. Dostupné z: https://access.redhat.com/site/documentation/en-S/Red_Hat_Enterprise_Linux/4/html/Reference_Guide/ch-kerberos.html#ftn.idp8334992

[29] LINN, J. *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures* [online]. Network Working Group, 1993 [cit. 2014-12-11]. 1421. Dostupné z: http://tools.ietf.org/html/rfc1421. Request for Comments.

[30] ČSN ISO/IEC 13888-3. *Informační technologie - Bezpečnostní techniky - Nepopiratelnost - Část 3: Mechanismy používající asymetrické techniky*. Česká technická norma (ČSN): Česká technická norma (ČSN), 1.5.2001. Dostupné z: http://csnonline.unmz.cz/Detailnormy.aspx?k=60916

[31] Midrot: Bugzilla. *Bugzilla.mindrot.org* [online]. 2014 [cit. 2014-12-11]. Dostupné z: https://bugzilla.mindrot.org/show_bug.cgi?id=2323

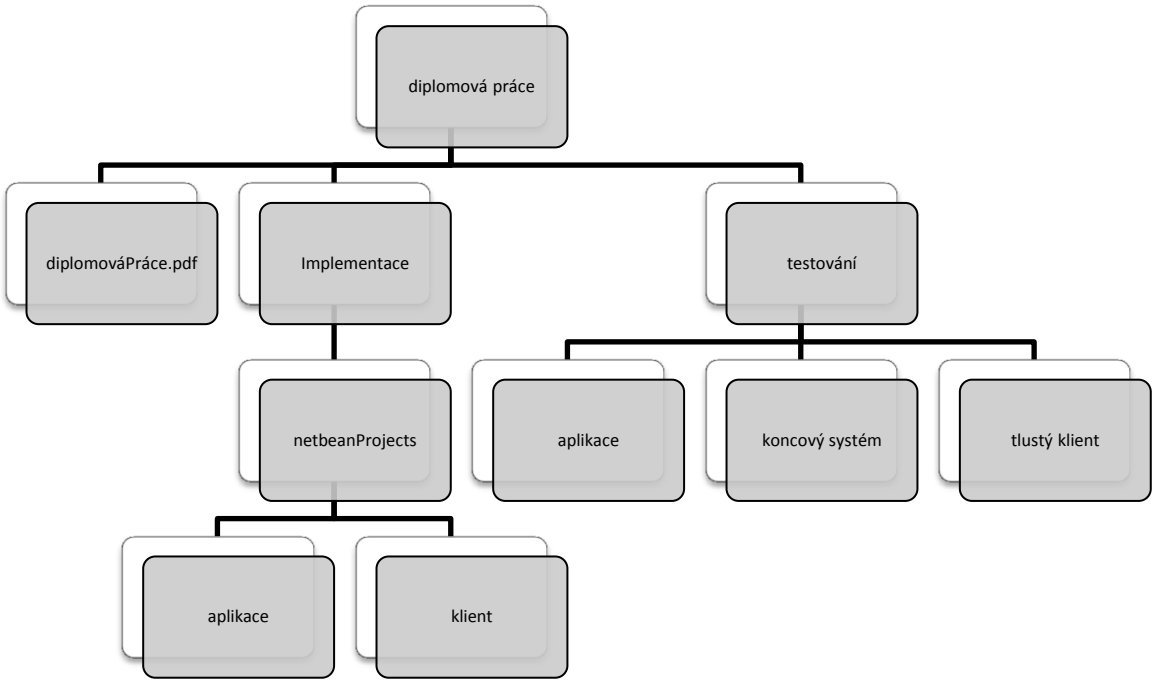
[32] LINN, J. *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures* [online]. Network Working Group, 1993 [cit. 2014-12-11]. 1421. Dostupné z: http://tools.ietf.org/html/rfc1421. Request for Comments.

[33] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile* [online]. Network Working Group, 2002 [cit. 2014-12-11]. 3208. Dostupné z: http://tools.ietf.org/html/rfc3280#section-3. Network Working Group.

[34] *The MD5 Message-Digest Algorithm* [online]. Network Working Group, 2002 [cit. 2014-12-11]. 1321. Dostupné z: https://www.ietf.org/rfc/rfc1321.txt. Network Working Group.

[35] *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography: Specifications Version 2.1* [online]. Network Working Group, 2002 [cit. 2014-12-11]. 3447. Dostupné z: https://www.ietf.org/rfc/rfc1321.txt. Network Working Group.

9 Obsah DVD



Hlavní adresář je rozdělen do dvou podadresářů „implementace“ a „testování“. V adresáři „implementace“ se nacházejí zdrojové kódy aplikace a tlustého klienta. Zdrojové kódy jsou zde vloženy jako projekty aplikace Netbeans.

Adresář testování obsahuje připravené virtuální stroje, ve formátu pro aplikaci VMware player, pro testování aplikace i koncového systému. V podadresáři „aplikace“ se nachází virtuální stroj nazvaný „Debian 7 64-bit“ s instalovanou aplikací. V podadresáři „koncový systém“ se nacházejí virtuální stroje s názvem „Ubuntu 64-bit“. Poslední podadresář obsahuje soubor typu „jar“ s kompilovaným kódem Tlustého klienta.