VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE EXTERNÍCH AUTENTIZAČNÍCH MODULŮ PRO NGINX

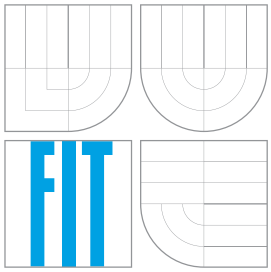
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETRA KAMENÍČKOVÁ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE EXTERNÍCH AUTENTIZAČNÍCH MODULŮ PRO NGINX

IMPLEMENT EXTERNAL AUTHENTICATION MODULES FOR NGINX

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETRA KAMENÍČKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. MAREK RYCHLÝ, Ph.D.

BRNO 2015

Abstrakt

Tato bakalářská práce se věnuje návrhu a vývoji autentizačních modulů pro webový server nginx, tak aby bylo možné aplikace a služby běžící na tomto serveru používat v rámci FreeIPA domény. V první části práce jsou vysvětleny základy architektur FreeIPA a serveru nginx a princip autentizace pomocí Kerberos a PAM. Druhá část řeší praktickou stránku problému - analýzu již existujícího řešení pro webový server Apache, návrh řešení pro nginx a popis postupu práce na vlastních modulech. V závěru jsou probrány možnosti konfigurace těchto modulů a návrhy na zlepšení.

Abstract

This bachelor's thesis describes the process of design and development of authentication modules for nginx web server. These modules are used for enrollment of nginx-based applications and services into FreeIPA environment. The first part of the thesis explains the basics of FreeIPA and nginx architectures and principles of Kerberos and PAM authentication mechanisms as well. The second part of the thesis solves the practical problems - existing Apache modules analysis, nginx design description and implementation details. The last part describes configuration requirements and possible enhancements.

Klíčová slova

autentizace, autorizace, PAM, nginx moduly, FreeIPA, Kerberos

Keywords

authentication, authorization, PAM, nginx modules, FreeIPA, Kerberos

Citace

Petra Kameníčková: Implementace externích autentizačních modulů pro nginx, bakalářská práce, Brno, FIT VUT v Brně, 2015

Implementace externích autentizačních modulů pro nginx

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana RNDr. Marka Rychlého, Ph.D. Další informace mi poskytl Jan Pazdziora z Identity Management týmu firmy Red Hat, který téma této bakalářské práce vypsal. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Petra Kameníčková
22. května 2015

Poděkování

Chtěla bych poděkovat vedoucímu mé bakalářské práce, panu Rychlému, za neuvěřitelnou trpělivost a podporu. Dále děkuji panu Pazdziorovi, který mi poskytl spoustu užitečných rad pro implementaci i teoretickou část.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V korporátní sféře je často pro správu poskytovaných síťových služeb používáno doménové prostředí. Domény jsou spravovány doménovými kontrolery, které poskytují jednotné rozhraní umožňující administrátorovi snadnou konfiguraci.

Z takových řešení je velmi rozšířený Active Directory od společnosti Microsoft. Ten správci domény umožňuje rychlou správu uživatelských účtů, nastavení přístupových práv a politik pro instalování aplikací. Zároveň poskytuje podporu pro Single Sign-On mechanismus, díky kterému se uživatel nemusí přihlašovat ke každé službě zvlášť. Stačí se přihlásit pouze jednou a ke všem dalším požadovaným službám proběhne přihlášení automaticky. Řešení, které Active Directory poskyuje, je sice komplexní, ale jeho bezproblémová konfigurace je zajištěna pouze v prostředí Windows.

Proto byl jako alternativa k Active Directory vyvinut produkt FreeIPA, který kombinuje řadu již existujících volně dostupných řešení a poskytuje jednotné rozhraní pro jejich konfiguraci i v prostředí systémů založených na Linuxu. Jedním z mnoha cílů FreeIPA je poskytnout možnosti správy a Single Sign-On i pro takové služby, které přímou podporu pro komunikaci s FreeIPA serverem nemají. A tady se dostáváme k hlavnímu důvodu vzniku této bakalářské práce.

V současné době existují moduly pro webový server Apache, kterými je možné FreeIPA řešení zpřístupnit i aplikacím a službám bez přímé podpory FreeIPA. Mezi takové moduly patří mod_authnz_pam, mod_auth_gssapi,
mod_lookup_identity a mod_intercept_form_submit. Díky nim je možné se autentizovat oproti adresářovému serveru integrovaném ve FreeIPA, a získat rozšířenou sadu informací o přihlášeném uživateli (členství ve skupinách, atributy…).

Apache ale začíná být postupně vytlačován jeho rychlejší alternativou - webovým serverem nginx. Z výše uvedených důvodu je nutné zajistit funkcionalitu zmíněných modulů i pro služby běžící právě na serveru nginx.

Úkolem této bakalářské práce je tedy seznámení s existujícím řešením pro webový server Apache a zajistit obdobu tohoto řešení pro nginx – tedy buď najít již existující moduly pro webový server nginx a navrhnout jejich správnou konfiguraci, nebo v případě, že modul ještě nebyl vyvinut, jej navrhnout a implementovat.

Kapitola 2

Základní pojmy z bezpečnosti

V této kapitole budou vysvětleny základní pojmy z oblasti počítačové bezpečnosti. Zejména nás zajímá názvosloví konceptu anglicky označovaného zkratkou AAA – authentication, authorization, accounting; v českém překladu autentizace, autorizace a účtování. Právě pojmy autentizace, autorizace a jejich správné pochopení jsou pro tuto bakalářskou práci velmi důležité.

2.1 Identita

Slovo identita používáme při porovnávání pojmů, objektů apod., které jsou zaměnitelné takovým způsobem, že mezi ně můžeme klást znaménko rovnosti. Identita osoby je pak definovaná jako „podmínka být sám sebou a nikým jiným“. [6]

Je také vhodné rozlišovat fyzickou a elektronickou identitu. Zatímco tu fyzickou máme pouze jednu - v reálném světě neexistuje nikdo, kdo by byl naše přesná kopie (DNA, vzhled, znalosti, vzpomínky...), elektronických identit si můžeme opatřit kolik chceme. V této práci se budeme věnovat výhradně identitám elektronickým – se zaměřením na identity uživatelů, strojů a služeb.

S pojmem identita také souvisí pojem identifikace, tedy proces určování identity.

2.2 Autentizace

Jako autentizaci označujeme proces zjišťování a ověřování identity nebo původu subjektu. V oblasti informačních technologií může být oním subjektem uživatel, program, nebo třeba i stroj. [7]

Na první pohled se může zdát, že autentizace je pouze jiný výraz pro identifikaci. Ale zatímco identifikace řeší úlohu „Kdo je to?“, autentizace hledá odpověď na otázku „Je opravdu tím, kým tvrdí, že je?“. Výsledkem autentizačního procesu je tedy nalezení/nenalezení a potvrzení/nepotvrzení konkrétní identity.

Ověřování identity uživatele může probíhat na základě různých faktorů, které mohou fungovat na různých principech:

- znalost informace – kombinace jména a hesla, PIN, odpověď na kontrolní otázku,
- vlastnictví – čipová karta, bezpečnostní tokeny, platební karta,
- existence – biometrické vlastnosti – DNA, obraz krevního řečiště, tvar ruky, tvar ucha, hlas,

- chování – dynamika podpisu, způsob chůze,
- lokace – IP adresa, bydliště,
- znalost nějaké osoby – řetězec důvěry.

V následujícím popisu se stručně seznámíme pouze s prvními třemi zmíněnými faktory, které považujeme za základní.

2.2.1 Autentizace znalostmi

Asi nejznámější a nejrozšířenější metoda autentizace funguje na principu znalosti tajné informace, kterou ostatní uživatele nemají k dispozici. Tímto tajemstvím může být kombinace uživatelského jména a hesla nebo třeba odpověď na kontrolní otázku.

Tento způsob autentizace je velmi jednoduchá na implementaci – postačí pouze databáze, proti které se porovnávájí informace získané od autentizujícího se uživatele.

Nejzranitelnější částí tohoto autentizačního faktoru je samotný přenos informací, případně uživatel samotný. Pro zvýšení bezpečnosti přenosu se používají různé mechanismy - SSL, Kerberos autentizace.

2.2.2 Autentizace vlastnictvím

Další ze základních faktorů funguje na principu vlastnictví. K autentizaci uživatele je třeba použít unikátní předmět, o kterém předpokládáme, že má uživatel u sebe. Nejčastěji to bývá softwarový nebo hardwarový token. V oblasti vlastnictví existují tři základní typy:

- statická informace, která není běžně viditelná – certifikát,
- synchronní dynamické heslo s omezenou platností – pomocí sesynchronizovaného času a klíče uživatel generuje heslo,
- asynchronní dynamické heslo s omezenou platností (systém výzva/odpověď) – server zašifruje výzvu a uživatel ji pomocí tokenu dešifruje a výsledek použije jako heslo.

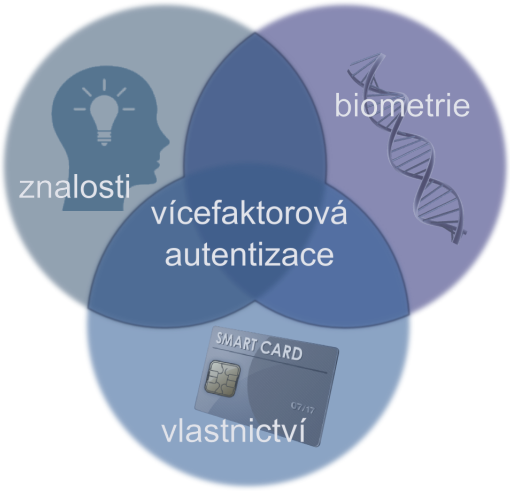
Mezi výhody používání tokenů patří absence statického hesla, a tedy i nutnost si jej pamatovat. V případě měnících se hesel je nespornou výhodou fakt, že i při nezabezpečeném spojení je odposlechnutné heslo víceméně k ničemu, protože má omezenou platnost. Dále pak skutečnost, že v případě ztráty nebo odcizení tokenu je pravděpodobné, že si toho uživatel všimne dřív než v případě prolomení statického hesla, a pravděpodobnost, že dojde ke skutečnému zneužití, je tedy menší.

2.2.3 Autentizace biometrickými vlastnostmi

Posledním hlavním faktorem je existence, tedy biometrické charakteristiky lidského těla. V této kategorii se k ověření identity uživatele používá biometrického měření (skenování krevního řečiště, porovnávání otisků prstů, autentizace hlasem...) a uživatel sám se tedy stává přístupovým klíčem.

2.2.4 Vícefaktorová autentizace

Jak je z předchozího popisu základních faktorů vidět, každý z nich má své klady a zápory. Jednotlivé faktory nedokážou samy o sobě často zajistit požadovaný stupeň bezpečnosti, a proto se v praxi používají jejich kombinace, kde výhoda jednoho faktoru dokáže snížit nebo zcela pokrýt nevýhody jiného.



Obrázek 2.1: Vícefaktorová autentizace

2.3 Autorizace

Autorizací rozumíme proces, při kterém ověřujeme, zda je subjekt oprávněn provádět jím požadovanou operaci, a na základě výsledku ověřovacího procesu mu povolíme nebo zamítneme přístup ke zdroji nebo k vykonání činnosti. Autorizace často úzce souvisí s procesem autentizace, ale existují i případy užití, kdy autentizace není před samotnou autorizací potřeba, např. autorizace anonymního uživatele na základě denní doby. Ověření oprávnění se provádí pomocí předem definovaných pravidel, která lze popsat různými modely řízení přístupu - např. přístupová matice, RBAC (Role Based Access Control), HBAC (Host Based Access Control).

2.4 Účtování

Účtování nebo také audit je činnost, při které se monitoruje aktivita uživatele. Toto sledování může zahrnovat množství přenesených dat, čas začátku a čas konce uživatelova působení v systému, procesorový čas zabraný programem, nebo třeba jaké konkrétní operace uživatel prováděl.

Získané informace se pak využívají ve statistických šetřeních a pomáhají například při vytváření nových pravidel přístupu a dalším plánování.

Kapitola 3

FreeIPA

FreeIPA (Free Identity Policy Audit) je open source projekt poskytující komplexní řešení pro centrální správu identit. Používá se jako doménový kontroler v prostředí, kde linuxové servery a klienti sdílí centrálně spravované služby.

FreeIPA kombinuje již existující open source technologie jako Linux, 389 Directory Server nebo MIT Kerberos 5, a poskytuje tak sjednocené prostředí a rozhraní pro správu domény. Součástí tohoto prostředí jsou jak konfigurační nástroje pro příkazovou řádku, tak i webové rozhraní.

Kromě adresářového serveru a autentizačního mechanismu Kerberos, který lze pro zvýšení bezpečnosti kombinovat s jednorázovými hesly, obsahuje FreeIPA i certifikační server Dogtag, webový server Apache, DNS server BIND a NTP server.

I přesto, že sám název FreeIPA napovídá, jaké jsou hlavní cíle projektu (IPA je v podstatě ekvivalentem AAA konceptu popsaného v předchozí kapitole), je FreeIPA momentálně zaměřena pouze na autentizaci a autorizaci. Účtovací část se pro svou komplikovanost prozatím odkládá a není vyloučeno, že z ní bude vytvořen samostatný projekt.

3.1 Historie

Projekt FreeIPA vznikl v roce 2007 pod záštitou firmy Red Hat. Hlavní motivací vývojářů byl fakt, že do té doby pro Linux dostupné autentizační technologie vyžadovaly příliš složitou konfiguraci a nebo nebyly kompatibilní.

První verze FreeIPA vyšla už v roce 2008 pod licencí GPL. Tato verze byla zaměřená hlavně na snadnou instalaci a správu FreeIPA serveru a základní správu identit uživatelů. Poskytovala nástroje pro automatizovanou konfiguraci adresářového a Kerberos serveru a snadnou replikaci. Webové rozhraní i nástroje příkazové řádky umožňovaly snadnou správu uživatelů a jejich skupin.

V roce 2011 byla vydaná FreeIPA verze 2. Tehdy byl do FreeIPA integrován DNS server a server certifikační autority. Přibyly také nové nástroje pro správu strojů, síťových skupin, správa přístupu na základě hostname, správa již vydaných certifikátů a keytabů, přidávání nové funkcionality pomocí pluginů.

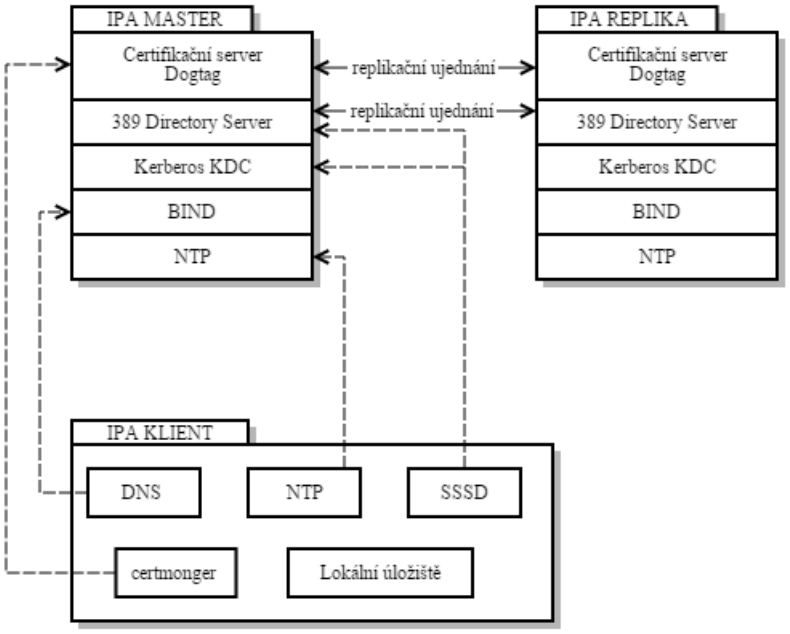
O rok později vyšla třetí verze. Nejzajímavější novinkou v této verzi byla možnost integrace s Active Directory pomocí Kerberos cross-realm trust mechanismu.

V současné době je aktuální verze 4 – ta představila možnost dvoufaktorové autentizace pomocí jednorázových hesel a Kerberos protokolu.

3.2 Architektura

První částí řešení FreeIPA jsou FreeIPA servery, které slouží jako úložiště informací o právech k přístupu a jako databáze identit uživatelů a strojů. Tyto servery také poskytují doménové služby jako Kerberos nebo DNS.

Master servery je možné klonovat a vytvářet tak repliky obsahující kopie záznamů z master serveru. Uzavřením tzv. replikačního ujednání mezi masterem a replikou se zajišťuje synchronizace těchto záznamů.



Obrázek 3.1: FreeIPA architektura

Kromě toho, že repliky slouží jako zálohy při výpadku master serveru, se také používají jako vyrovnávače zatížení doménového master serveru. Toto vyrovnávání se dělá automaticky nastavováním priority SRV záznamů na serveru DNS, která určuje v jakém pořadí má klient servery kontaktovat.

Další částí jsou FreeIPA klienti. Z pohledu uživatele je FreeIPA klient stroj, který uživateli poskytuje v rámci domény nějakou službu.

Klientem se může stát jakýkoli stroj existující v rámci FreeIPA domény, který je nakonfigurovaný tak, aby používal služby poskytované FreeIPA serverem. Ve FreeIPA prostředí se pro tuto konfiguraci používá tzv. enrollment mechanismus, který automaticky nainstaluje a nakonfiguruje všchny potřebné služby potřebné pro kooperaci s FreeIPA serverem. Kromě toho se na klientovi vytvoří lokální úložiště (LDB databáze nebo XML soubory) sloužící k dočasnému ukládání informací získaných z FreeIPA serveru. To napomáhá ke snížení režie mezi serverem a klientem, a tedy i zvýšení výkonu celého řešení. [1]

3.3 Kerberos

Kerberos je síťový protokol používaný pro autentizaci a autorizaci fungující na principu dočasných tiketů. Obecně je tento protokol považovaný za bezpečný (v rámci komunikace jako takové). Omezená platnost tiketu alespoň částečně chrání proti odposlechnutí a následnému použití stejných rámců. A díky použití samotných šifrovaných tiketů se heslo, ani jeho hash nikdy nepřenáší nezabezpečenou sítí.

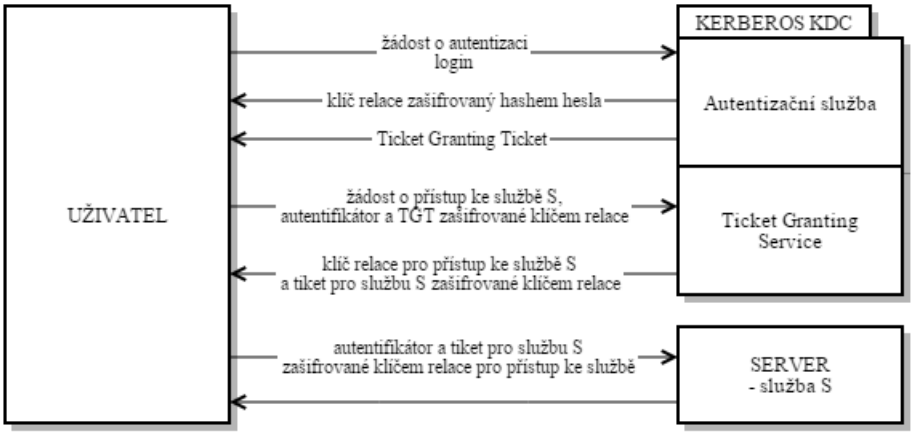
Kerberos používá k autentizaci faktor znalostí, ale pro zvýšení bezpečnosti se často uživatelské heslo kombinuje s používáním jednorázových hesel generovaných softwarovými nebo hardwarovými tokeny.

3.3.1 Historie

Kerberos byl navržen koncem osmdesátých let minulého století na Massachusettském technologickém institutu. První verze sloužily pouze pro interní potřeby univerzity, ale od verze 4 se začal rozšiřovat i do běžného prostředí. V dnešní době se používá pátá verze tohoto protokolu, jež byla do současné podoby upravena v roce 2005. Tato verze je popsaná specifikací RFC 4120 [5].

3.3.2 Princip

Klíčovou částí mechanismu je KDC (Key Distribution Center), které poskytuje dvě služby: autentizační službu a TGS (Ticket-Granting Service).



Obrázek 3.2: Průběh Kerberos autentizace

Autentizační služba provádí autentizaci uživatele. Na základě obdrženého uživatelského jména vyhledá heslo uložené v databázi a jeho hashem zašifruje klíč relace. Zároveň také sestaví TGT (ticket-granting tiket), který obsahuje informace o uživateli (ID, IP adresa…), informace o TGS, různé příznaky, dobu platnosti tiketu a klíč relace. TGT je pak zašifrovaný klíčem, který je známý pouze autentizační službě a TGS. Klíč relace a TGT jsou poté autentizační službou odeslány zpět klientovi, který prokáže svojí identitu (v prostředí protokolu Kerberos se častěji používá pojem principal) tím, že dokáže dešifrovat zprávu s klíčem relace.

Klient následně kontaktuje TGS, kterému odešle TGT , žádost o přístup ke službě a autentikátor zašifrovaný klíčem relace. Autentikátor obsahuje informace o uživateli a časové razítko. TGS na základě porovnání informací z TGT a autentikátoru ověří uživatele a platnost tiketu. A následně nazpět odešle ticket a klíč pro přístup k požadované službě, které jsou zašifrovány klíčem relace.[5]

Celý tento proces funguje jako SSO (Single Sign-On). To znamená, že na klientské straně uživatel vloží heslo pouze jednou a veškerá další autentizace, případně autorizace, se děje automaticky pomocí tiketů.

3.3.3 GSSAPI

GSSAPI (Generic Security Services Application Program Interface) tvoří standardizované aplikační rozhraní autentizace mezi klientem a serverem. K výměně dat jsou používány tokeny zvyšující bezpečnost komunikace. Princip GSSAPI mechanismu je detailně popsán v RFC 2743[1] a RFC 2744[2]. Protože GSSAPI sám o sobě nepopisuje způsob, jakým se navazuje spojení mezi účastníky autentizačního procesu, byl zaveden další standard, SPNEGO.

SPNEGO[3] (definovaný RFC 4178[4]) popisuje způsob komunikace při vyjednávání autentizační metody. Toho se využívá v případě, že klient nebo server neví, jakou autentizační metodu používá druhá strana. Pomocí SPNEGO dojde k výměně seznamu dostupných autentizačních mechanismů a vybere se ten nejvhodnější.

Oba tyto mechanismy jsou velmi často používány právě pro autentizaci pomocí Kerberos protokolu.

3.4 389 Directory Server

389 Directory Server je open-source adresářový server vyvíjený v rámci Fedora komunity. Projekt vznikl jako odnož původního slapd, ze kterého se později vyvinulo známější řešení OpenLDAP.

Mezi těmito projekty existuje jen velmi málo rozdílů. Oba si zakládají na svojí otevřenosti, na vysokém výkonu, poskytují možnost multi-master replikace, zajišťují bezpečnout autentizaci a přenos pomocí TLSv1 a SASL a podporují LDAPv3.

3.5 SSSD

SSSD (System Security Services Daemon) je systémová služba, jejíž primárním účelem je poskytnutí rozhraní pro přístup ke vzdáleným úložištím identit a autentizačních údajů v rámci více domén – v našem případě je tímto úložištěm právě FreeIPA.

Framework, který SSSD poskytuje, umožňuje dočasné lokální ukládání dat, takže i klient bez přístupu k Internetu může používat služby jako FreeIPA nebo LDAP. SSSD také obsahuje rozhraní pro komunikaci s PAM (viz kapitola 4) a NSS[5].

[1] http://tools.ietf.org/html/rfc2743
[2] http://tools.ietf.org/html/rfc2744
[3] The Simple and Protected GSSAPI Negotiation Mechanism
[4] http://tools.ietf.org/html/rfc4178
[5] Name Service Switch

Kapitola 4

PAM

PAM[1] je sada knihoven poskytující univerzální aplikační rozhraní pro různé způsoby autentizace. Umožňuje aplikacím, aby byly zcela nezávislé na autentizačních metodách. V případě nutnosti přidání nové metody tedy vývojář nemusí nijak zasahovat do kódu aplikace; stačí pouze změnit konfiguraci PAM.

Koncept PAM byl představen už v roce 1995 společností Sun Microsystems. Hlavní motivací pro vývoj PAM byla snaha oddělit aplikaci od autentizačního mechanismu. Do té doby bylo při vývoji každé nové aplikace nutné znovu reimplementovat všechny potřebné mechanismy pro autentizaci uživatele, čímž se do aplikací zanášely zbytečné bezpečnostní chyby. [4] Kromě snadného nahrazení používané autentizační metody, umožňuje PAM použití i více autentizačních možností zároveň a mapování služeb, uživatelských jmen a autentizačních tokenů skrze různé autentizační domény.

4.1 Princip

PAM se skládá z knihovny pro rozhraní, které implementuje API (Application Programming Interface), a mnoha autentizačních modulů, které jsou dynamicky linkované a volané SPI (Service Provider Interface).

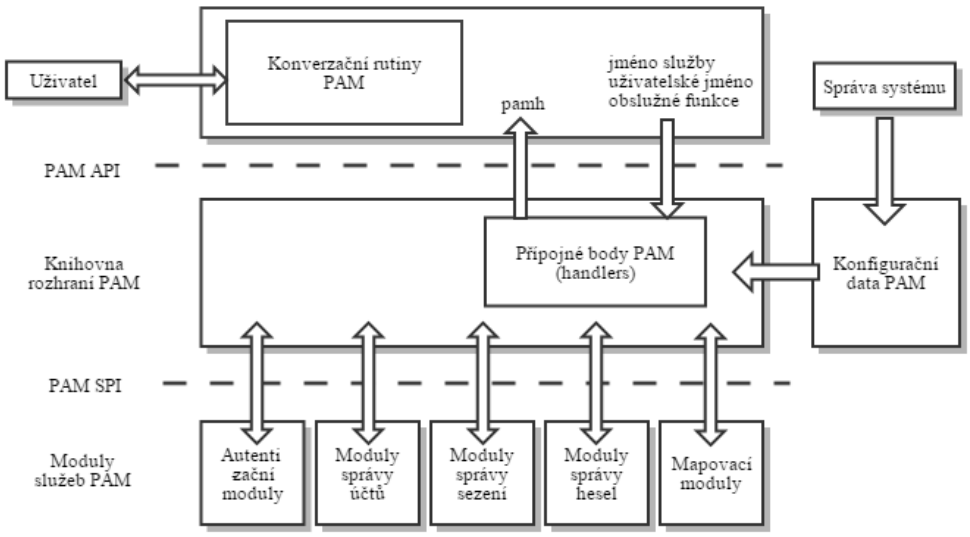
K vytvoření PAM sezení se používá funkce `pam_start()`. Voláním této funkce aplikace (webový server) předá jméno služby, jméno uživatele, a ukazatel na strukturu s informacemi potřebnými ke komunikaci; tato struktura obsahuje mimo jiné i heslo uživatele. Návratovou hodnotou této funkce je pak ukazatel na vnitřní PAM strukturu (v obrázku 4.1 označenou jako `pamh`) používanou pro další volání. Během sezení je možné volat další funkce PAM knihovny, ale pro potřeby této práce postačí, když se seznámíme s funkcemi pro autentizaci (`pam_authenticate()`) a autorizaci(`pam_acct_mgmt()`).

Funkce `pam_authenticate` se používá pro ověření identity uživatele. Jako parametry jsou jí předávány ukazatel na PAM strukturu pamh a příznaky, kterými se nastavuje další chování.

Funkce `pam_acct_mgmt()` je volána pro ověření platnosti uživatelského účtu. Kontroluje autentizační token, platnost účtu a ověřuje omezení přístupu. Jsou jí předávány tytéž parametry jako funkci `pam_authenticate()` - tedy ukazatel na strukturu pamh a příznaky pro nastavení chování PAM.

Poslední funkcí, kterou budeme v této práci potřebovat, je `pam_end()`. Voláním této funkce se ukončí sezení vytvořené funkcí `pam_start()`. Jako vstupní parametr je očekáván

[1] Pluggable Authentication Modules



Obrázek 4.1: Schéma PAM architektury

ukazatel na strukturu pamh a návratový kód posledního volání PAM.

4.2 Konfigurace

Nastavení PAM se provádí pomocí konfiguračních souborů pro jednotlivé služby, které bývají v linuxových distribucích uloženy v adresáři `/etc/pam.d/`.

PAM poskytuje čtyři nezávislé funkční skupiny:

- autentizaci uživatele (auth),
- kontrolu přístupu (account),
- správu relace (session),
- změnu hesla (password).

Skupina modulů auth provádí klasickou autentizační činnost dle zvoleného modulu. Nastavení account modulů umožňuje kontrolu přístupu a ověřuje aktuálnost hesla a dalších informací. Moduly typu session se aktivují před a po provedením služby a obvykle nastavují proměnné prostředí, provádí chroot nebo protokolují. A typ modulů password umožňuje správu hesla a kontrolu, zda byly splněny podmínky pro heslo - jeho délka, výskyt speciálních znaků atd.

Počet modulů pro jednotlivé skupiny není nijak omezen, pořadí jejich zpracování pak závisí na pořadí uvedeném v konfiguračním souboru pro službu. Kromě toho je každému modulu přiřazen kontrolní příznak, který určuje zda je modul povinný, volitelný a v jaké fázi zpracování ohlašuje chybu.

Obsah konfiguračního souboru `/etc/pam.d/service`, pro autentizaci a autorizaci služby `service` skrz SSSD, by pak vypadal následovně:

```
   auth     required     pam_sss.so
account     required     pam_sss.so
```

Kapitola 5

nginx

nginx je open source webový server a reverzní proxy podporující protokoly HTTP, HTTPS, SMTP, POP3 a IMAP. Už od svého vzniku je zaměřený na vysoký výkon, souběžné zpracovávání požadavků s nízkou náročností na systémové prostředky.

nginx byl implementován v jazyce C a je od základu původní (resp. není založen na žádném jiném projektu). Až na drobné výjimky (zlib, PCRE, OpenSSL), které je možné v případě potřeby vynechat, používá výhradně vlastní moduly a knihovny. Díky tomu je nginx podporován mnoha architekturami a operačními systémy. V současné době je dostupný pro systémy na bázi Linuxu, FreeBSD, Solaris, Mac OS X, AIX a Microsoft Windows.

I přesto, že má nginx široké využití - používá se jako reverzní proxy, webový server, HTTP cache nebo vyrovnávač zátěže (load balancer) - budeme ho v rámci této práce používat výhradně jako webový server.

5.1 Historie

Vývoj serveru nginx započal v roce 2002 Igor Sysoev. Ten původně vyvíjel proxy modul (mod_accel) pro webový server Apache, ale během jeho implementace narazil na problémy spojené s omezenou škálovatelností a výkonem serveru Apache, a rozhodl se pro vývoj vlastního webového serveru.

První verze nginxu vyšla v roce 2004 pod dvouklauzulovou BSD licencí kompatibilní s GNU GPL. Původně byla první verze zamýšlena pouze jako doplněk k Apache serveru, který by zpracovával statický obsah jako HTML, CSS, JavaScript a obrázky, a ulehčil tak zátěži Apache serveru, ale později byl nginx rozšířen o další funcionalitu, která z něj udělala plnohodnotný webový server – např. integrace částí umožňující zpracovávání protokolů FastCGI, uswgi nebo SCGI.

5.2 Rozšíření

Podle posledního průzkumu společnosti Netcraft z dubna 2015 [1], hostuje nginx na více než 125 milionů webových stránek, což tvoří téměř 15 % z celkového počtu všech webů. To nginx řadí na třetí místo nejpoužívanějších webových serverů hned za Apache a Microsoft Server. V oblasti aktivních webů obsadil nginx druhé místo. Používá ho více než 25 milionů aktivních stránek, tedy asi 14 % z celkového počtu aktivních webů. V rámci aktivních webů

[1] `http://news.netcraft.com/archives/2015/04/20/april-2015-web-server-survey.html`

je to druhý nejrozšířenější webový server. První místo obsadil opět Apache, který používá více než 50 % aktivních webů.

O širokém použití serveru nginx vypovídá i fakt, že ho používají tak známé služby jako Netflix, Pinterest, Instagram, GitHub nebo třeba WordPress.

5.3 Architektura

Jak už bylo zmíněno v úvodu, nginx je velmi zaměřený na souběžné zpracovávání s nízkou náročností na paměť. Byl navržen tak, aby dokázal spolehlivě vyřešit tzv. C10K problém[2] - tedy zpracování 10 tisíc souběžných požadavků.

Během tradičního vícevláknového zpracování, které používá např. server Apache, dochází při velkém množství persistentních spojení k poměrně rozsáhlé alokaci paměti. Pro každého připojeného klienta je potřeba vytvořit samostatné vlákno, které si běžně alokuje více než 1 MB paměti (v závislosti na požadavku).

V případě, že má tedy server, který používá vícevláknový přístup, současně obsloužit 10 tisíc klientů, bude muset alokovat více než 10 GB operační paměti. To může vést k vyčerpání dostupné operační paměti a odkládání dat na pevný disk, čímž se rapidně sníží výkon serveru. Kromě toho, v případě dosažení maximálního počtu otevřených spojení pak server začne nově příchozí požadavky odmítat. Díky tomu jsou servery používající vícevláknový přístup také více náchylné k Slowloris nebo podobným DoS útokům.

Proto se nginx inspiroval mechanismy uplatňovanými v operačních systémech a použil modulární, událostmi řízenou, asynchronní, jednovláknovou, neblokující architekturu. Ke zpracování úkonů spojených se správou síťových spojení a získávání obsahu, používá nginx události a řadu optimalizovaných metod pro vstupní a výstupní operace jako kqueue, eventport nebo epoll. Díky tomuto nginx potřebuje ke zpracování 10 tisíc požadavků (viz výše) alokovat paměť pouze v řádech několika desítek až stovek MB.
Konkrétní porovnání paměťové náročnosti serverů Apache a nginx je možné najít např. zde: `http://wiki.dreamhost.com/Web_Server_Performance_Comparison`.

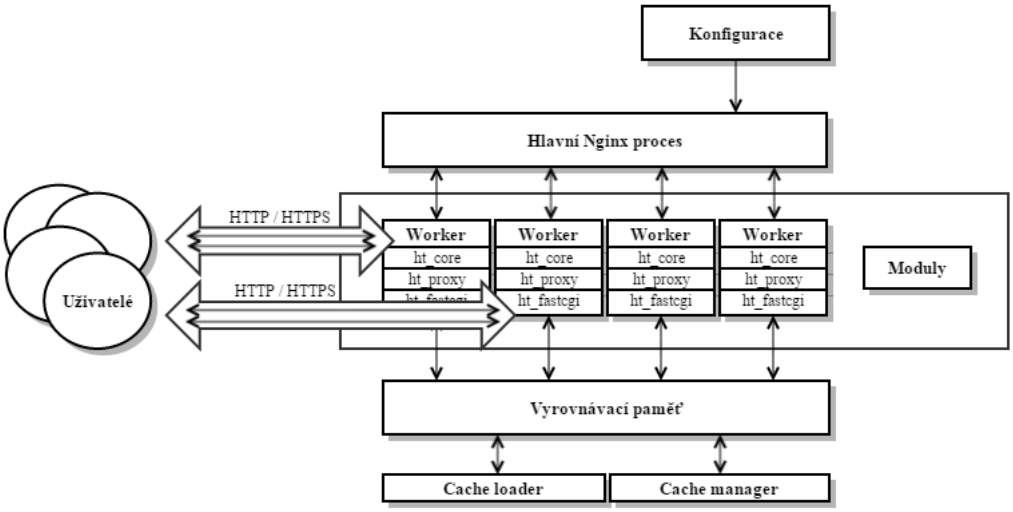
nginx sestává z několika jednovláknových procesů, které pro meziprocesovou komunikaci používají sdílenou paměť:

- hlavní (master) proces,
- worker procesy,
- speciální procesy spravující vyrovnávací paměť (cache loader, cache manager).

Hlavní proces je zodpovědný za kontrolu konfigurace a případnou rekonfiguraci. Mimo to také spravuje sockety a obstarává běh několika worker procesů, logování a práci s Perl skripty. Úkolem worker procesů je přijímat a zpracovávat spojení a požadavky, poskytovat revezní proxy, filtry a obecně zajišťovat veškerou další funkcionalitu, kterou nginx nabízí. Procesy řídící vyrovnávání paměti kontrolují data uložená na disku, doplňují metadata do databáze uložené ve sdílené paměti a sledují platnost těchto metadat.[2]

Dále se budeme podrobněji věnovat hlavně worker procesům (resp. jejich modulům), protože právě ty pracují s moduly jako takovými.

[2] `http://www.kegel.com/c10k.html`



Obrázek 5.1: Schéma architektury serveru nginx

5.3.1 Worker procesy

Jak je z předchozího popisu patrné, hlavní proces nginx serveru nevytváří nového workera pro každé spojení. Místo toho je během inicializace vytvořeno jen tolik worker procesů, kolik udává konfigurační direktiva `worker_processes`. V praxi se běžně používá tolik worker procesů, kolik má server k dispozici jader procesoru.

Každý z těchto procesů pak obstarává tisíce příchozích požadavků, které přichází na sdílený socket. Rozdělování požadavků mezi worker procesy obstarávají mechanismy jádra operačního systému. Díky jen několika málo předem vytvořeným procesům se výrazně šetří operační paměť potřebná pro obsloužení tisíců klientských požadavků. Zároveň se tím, že se pro každý požadavek nevytváří zvláštní proces nebo vlákno, také šetří strojový čas, který by byl potřeba pro vytvoření a ukončení vlákna.

Každý worker proces se skládá z jádra a modulů, které musí být zkompilovány společně. Jádro workera se stará o síť a aplikační protokoly. V jádře běží nekonečný cyklus, který volá registrované moduly zodpovědné za obsloužení právě probíhající fáze zpracování požadavku. Postupné volání modulů odvádí veškerou práci na prezenční a aplikační vrstvě síťového modelu: čtení a zápis ze sítě nebo úložiště, filtrování, upravování a skládání obsahu odpovědi, předávání požadavků dál (proxy) atd.

Cyklus ve worker procesu řídí funkce `ngx_worker_process_cycle()`. Ta se skládá zhruba z následujících částí. Nově příchozí událost se zpracuje mechanismy konkrétního operačního systému – epoll, kqueue atd. Událost je poté přijata ke zpracování worker procesem. Postupně je zpracována hlavička a tělo požadavku. Následně se zavolá obslužná rutina dle konfigurace pro daný kontext (viz kapitola 5.4) a požadavek pak prochází postupně všemi fázemi zpracování (viz kapitola 5.3.2). Tím je vytvořen odpovídající obsah odpovědi, který se odešle se klientovi. Zpracování požadavku je tímto u konce a zbývá jen inicializovat časovače a události.

Jak bylo zmíněno výše, nginx používá pro meziprocesovou komunikaci sdílenou paměť, přesněji řečeno paměťové bazénky (memory pool). Paměťový bazének je v podstatě sekvence předalokovaných bloků na haldě. Bazénky jsou používány pro ukládání vyrovnávacích me-

tadat, relačních dat SSL a dalších informací potřebných pro zpracování požadavku. Mezi výhody jejich používání patří snížení režie alokace a dealokace paměti pro objekty a snížení rizika úniku paměti.

5.3.2 Moduly

V nginx serveru mohou moduly zastupovat jednu ze tří rolí:

- obslužná rutina (handler), která zpracovává požadavek a produkuje výstup,
- filtr - upravuje výstup obslužných rutin (komprese, úprava hlaviček),
- vyrovnavač zátěže (load-balancer), který vybírá koncový server a přeposílá mu požadavek.

Během startu serveru se každá obslužná rutina registruje do konkrétní lokace definované v konfiguračním souboru. Registrace se provádí přidáním ukazetele na obslužnou rutinu do pole ukazatelů. Pokud je pro jednu lokaci takových rutin více, může docházet ke konfliktu. Pak závisí na další konfiguraci (a konkrétní fázi), jestli budou volány i další obslužné rutiny nebo pouze první nalezená. Například při použití direktivy satisfy pro fázi přístupu se dle parametru direktivy volají buď všechny obslužné rutiny (satisfy all), nebo se volání ukončí u první úspěšné (satisfy any). Volání obslužných rutin se provádí v opačném pořadí než bylo pořadí jejich registrace.

Pokud se ukáže, že obslužná rutina má přeposlat požadavek jinam, volají se moduly vyrovnávající zátěž. nginx poskytuje dvě metody rozhodování, který z koncových serveru pro vyrovnávání bude použit:

- round-robin, přeposílající požadavky na servery rovnoměrně,
- IP hash metodu, která zajišťuje, že požadavky konkrétního klienta dojdou vždy na tentýž server.

V případě že obslužná rutina neskončí chybou, jsou volány filtry, jejichž počet není pro jednu lokaci nijak omezen. Pořadí, v jakém jsou prováděny, je určeno během kompilace serveru. Filtry při zpracovávání dat používají vyrovnávací paměť. Díky tomu je možné, aby byla odpoveď zpracovávána dřív, než poslední filtr dokončí svojí činnost. [3]

Běžný postup zpracování HTTP požadavku z pohledu modulů tedy vypadá zhruba následovně. Klient pošle serveru HTTP požadavek. Jádro nginx serveru vybere obslužnou rutinu pro danou fázi (phase handler) na základě konfigurace odpovídající požadavku. V případě nakonfigurovaného vyvažování zátěže se zvolí cíl přesměrování. Rutina obsluhující fázi pak odvede svojí práci a pošle výstup filtrům. Po zpracování všemi filtry se konečně výsledná odpověď pošle klientovi.

Každý modul je nutné zaregistrovat do některé fáze zpracování požadavku. Těchto fází nabízí nginx celkem deset - resp. méně, protože ne všechny fáze zpracování jsou určeny pro přidání vlastního modulu. Po přečterní hlavičky požadavku se vyhledá odpovídající konfigurace virtuálního serveru. Pokud je server nalezen, požadavek je postupně zpracován následujícími fázemi:

1. fáze přepisování URI (server rewrite phase) – během této fáze dochází k úpravě URI na úrovni virtuálního serveru; pokud se cyklus přepisování URI opakuje víc než desetkrát, server vrátí HTTP stavový kód 500 – Internal Server Error,

2. konfigurační fáze (config phase) – v této fázi se načte konfigurace pro lokaci; zároveň je vyplněna položka Location v hlavičce odpovědi na požadavek,

3. přepisovací fáze (rewrite phase) – probíhá podobně jako první fáze; tentokrát na úrovni lokace,

4. popřepisovací fáze (post rewrite phase) – provádí akce dokončující přepisovací fázi,

5. předpřístupová fáze (preaccess phase) – fáze spouštěná před uplatňováním omezení přístupu; v této fázi se uplatňují direktivy jako degradation (odmítnutí přístupu při nízkých systémových prostředcích),

6. přístupová fáze (access phase) - v této fázi jsou volány autentizační moduly; kromě kontorly přístupu na základě např. IP adresy (direktivy allow, deny) jsou zde také spouštěny další autentizační moduly, např. moduly zajišťující HTTP Basic autentizaci, Kerberos nebo PAM,

7. popřístupová fáze (post access phase) – zajišťuje kontrolu předchozí fáze; v podstatě slouží pouze pro kontrolu dle direktivy satisfy,

8. fáze pro zpracování `try_files` direktivy (try_files phase) – kontroluje existenci souborů dle direktivy try_files

9. fáze generování obsahu (content phase) – v této fázi se spouští další moduly vytvářející obsah odpovědi – např. FastCGI modul,

10. logovací fáze (log phase) – zpracování informací a jejich zaznamenání do logovacích souborů.

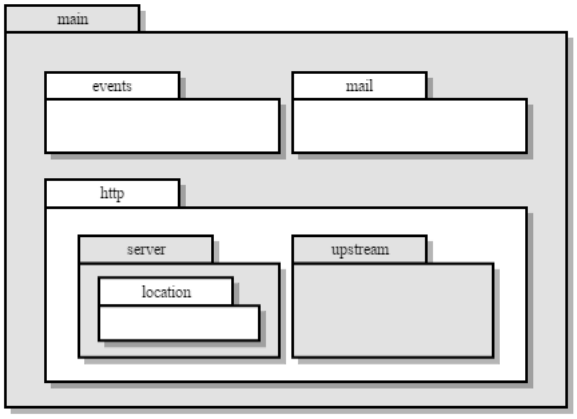
Do konfigurační, popřístupové, popřepisovací a fáze pro zpracování try_files direktivy není možné přidávat žádné vlastní moduly, resp. ukazatele na obslužné rutiny.[3]

Obslužná rutina může skončit různými způsoby – kromě úspěšného a neúspěšného zpracování může také požadavek odmítnout (ideálně by měl vývojář zajistit, aby modul vrátil odmítnutí co nejdříve po tom, co je zjištěno, že modul nemá jak požadavek zpracovat). V případě, že modul odmítl požadavek zpracovat se přistupuje k výchozím obslužným rutinám.

[3] `http://www.nginxguts.com/2011/01/phases/`

5.4 Konfigurace

Konfigurační soubor nginx serveru definuje chování serveru pro jednotlivé oblasti nazývané kontexty. Do každého z kontextů je možné přiřadit direktivy, které popisují konfiguraci jednotlivých modulů.



Obrázek 5.2: Hierarchie nginx kontextů

Kontexty tvoří stromovou strukturu, díky které je zajištěna dědičnost konfigurace mezi jednotlivými úrovněmi. Pořadí direktiv v konfiguračním souboru neovlivňuje pořadí jejich zpracování. To, v jakém sledu budou moduly a jejich funkce zavolány, záleží pouze na pořadí registrace jednotlivých modulů a na jejich vnitřní struktuře. V každém z modulů je tedy třeba definovat funkce, které se volají pro zpracování konfigurace jednotlivých kontextů.

Kapitola 6

Analýza problému

V této kapitole se detailněji seznámíme s implementací již existujících modulů pro webový server Apache a prozkoumáme možnosti konfigurace stávajících řešení.

Moduly webového serveru Apache jsou narozdíl od modulů serveru nginx linkovány dynamicky. K jejich registraci do procesu zpracovávajícího požadavky se používají tzv. háky (hooks). "Zaháknutí"se provádí pomocí volání různých funkcí, kde každá umožňuje registraci do jiné fáze zpracování požadavku (tyto fáze jsou odlišné od těch, co používá server Nginx (viz kapitola 5.3.2)).

Každý modul musí mít definovanou strukturu AP_MODULE_DECLARE_DATA. Tato struktura obsahuje ukazatele na funkce obstarávající konfiguraci a ukazatel na registrační funkci pracující s háky. Jméno této struktury se používá při konfiguraci Apache serveru. Konkrétně jej používá direktiva LoadModule. Ta jako parametry bere právě název struktury a umístění dynamicky linkované knihovny modulu.

6.1 mod_auth_kerb

Mod_auth_kerb[1] je modul pro webový server Apache poskytující podporu pro Kerberos autentizaci. Modul poskytuje metody pro použití Kerberos protokolu verze 5, i starší verze 4. Zároveň podporuje jak MIT verzi protokolu, tak i její implementaci Heimdal. Kromě autentizace Kerberos protokolem jako takové, obsahuje podporu pro SPNEGO s GSSAPI. Díky tomu je podporován mechanismus HTTP Negotiate[2] umožňující použití Kerberos autentizace v prohlížečích.

Kromě Negotiate metody je možné zapnout podporu Basic autentizace, která se při selhání vyjednávání použije jako záložní mechanismus. Basic autentizační mechanismus ale není nijak zabezpečen; během přenosu se používá snadno prolomitelné kódování Base64. Proto je při použití Basic autentizace je doporučeno použít další modul, který zajistí bezpečný přenos informace, např. mod_ssl. Mod_auth_kerb podporuje webový server Apache verze 1.x i 2.x a vyžaduje knihovnu Kerberos verze 1.4.0 nebo vyšší.

Mod_auth_kerb je postupně nahrazován novým modulem - mod_auth_gssapi[3], jenž plně podporuje SPNEGO a používá pouze GSSAPI volání. Pro správnou funkčnost je vyžadován server Apache verze 2.4 a vyšší. Dále je doporučeno mít nainstalovanou knihovnu krb5 verze

[1] `http://modauthkerb.sourceforge.net/`

[2] `https://www.ietf.org/rfc/rfc2295.txt`

[3] `https://github.com/modauthgssapi/mod_auth_gssapi`

1.11 a vyšší. Dále se budeme věnovat pouze modulu mod_auth_kerb.

6.1.1 Konfigurace

Abychom zjistili, jaké funkce modulu poskytuje, seznámíme se s jeho konfiguračními direktivami. V rámci tohoto dokumentu nám postačí představit pouze základní direktivy pro nastavení autentizace přes Kerberos verze 5. Informace o dalších direktivách je možné nalézt na stránkách projektu (viz poznámka pod čarou).

`AuthType typ`

Direktiva AuthType nastavuje metodu použitou pro autentizaci uživatele. Podporované výchozí typy (None, Basic (mod_auth_basic), Digest (mod_auth_digest) a Form (mod_auth_form) je možné snadno rozšířit. Pomocí funkce ap_auth_type() lze zjistit jaký parametr byl touto direktivou předán. Modul mod_auth_kerb zpracovává parametry "Kerberos", "KerberosV4"a "KerberosV5".

`AuthName jméno`

Parametr direktivy AuthName určuje oblast (realm) autentizace. Jméno oblasti se používá například v hlavičce odpovědi (položka WWW-Authenticate) a je použito prohlížečem při sestavení dialogového okna pro Basic autentizaci.

`Require uživatel`

Určuje způsob autorizace autentizovaného uživatele. Lze takto přesně nadefinovat uživatele nebo skupiny mající práva k přístupu. Také je možné použít parametr "valid-user". Tím je povolen přístup každému autentizovanému uživateli.

`KrbMethodNegotiate on|off`

Zapíná a vypíná použití Negotiate metody.

`KrbMethodK5Passwd on|off`

Nastavuje Basic autentizaci po selhání Kerberos autentizace.

`KrbAuthRealms oblast1 [oblast2 ... oblastN]`

Definuje Kerberos oblast (realm), která bude použita pro autentizaci. Konfigurace pro každou oblast (např. adresa KDC serveru) se získává z konfiguračního souboru pro Kerberos. V linuxových distribucích většinou jde o soubor `/etc/krb5.conf`.

`KrbServiceName služba`

Určuje jméno služby, pro kterou Kerberos konfigurujeme. Jméno musí korespondovat s tím, co je uvedené v keytabu.

`Krb5Keytab /cesta/ke/keytabu`

Specifikuje umístění používaného Kerberos keytabu. Apache musí mít nastavená práva pro čtení tohoto souboru.

6.1.2 Vnitřní struktura

Nejprve dochází k zahákování modulu do části podobné přístupové fázi serveru nginx. Jako obslužná funkce je registrována `kerb_authenticate_user()`.

Funkce `kerb_authenticate_user()` přijímá jako parametr ukazatel na strukturu požadavku typu `request_rec`. Jako první je načtena a zpracována konfigurace modulu. Dochází ke kontrole parametru předaného direktivou `AuthType`. V případě, že je parametr určen pro tento modul a požadavek tedy nebyl odmítnut, jsou postupně zvalidovány další konfigurační direktivy. Následně jsou zpracovány údaje z hlavičky HTTP požadavku.

Dále je zkontrolován stav direktivy `KrbMethodNegotiate` - v případě, že byla nastavena, dochází k získání autentizčních údajů voláním funkce `get_gss_creds()`. Poté jsou postupně volány funkce rozhraní GSSAPI. Vzhledem k tomu, že alternativa tohoto modulu už pro server nginx existuje, není potřeba přesně popisovat, k jakým konkrétním voláním dochází.

V případě, že `KrbMethodNegotiate` nebyla použita, probíhá kontrola direktivy `KrbMethodK5Passwd`. Při jejím nastavení je volána funkce `authenticate_user_krb5pwd()`. Ta dekóduje přihlašovací údaje, načte keytab soubor a nakonec volá funkci `verify_krb5_user()`, která provede akce popsané v kapitole 3.3.

6.2 mod_authnz_pam

Modul mod_authnz_pam[4] umožňuje použití PAM rozhraní pro autentizaci a autorizaci uživatele. Kromě toho poskytuje autentizaci a autorizaci údajů získaných pomocí Basic autentizace. Primárním účelem tohoto modulu pro potřeby FreeIPA je zajištění autentizace přes SSSD - resp. pam_sss.so modulem.

6.2.1 Konfigurace

Modul lze použít dvěma způsoby. První z nich používá PAM pouze jako autorizační podporu pro jiný autentizační mechanismus - např. autentizace uživatele přes Kerberos, ověření práv pro přístup oproti FreeIPA serveru.
PAM modul lze použít i jako autentizační a autorizační doplněk k Basic autentizaci. Tento způsob je podporován pouze u verze Apache serveru 2.4 a vyšší.

```
Require pam-account jméno_služby
```

Pokud chceme používat PAM pouze pro autorizaci, je nutné nastavit Apache direktivu `Require`. Ta běžně slouží ke specifikaci autorizačních podmínek. Modul mod_authnz_pam podporuje parametry "pam-account"a jméno služby, oproti které je PAM nakonfigurován.

```
AuthBasicProvider PAM
```

Nastavuje ověřování Basic autentizace proti PAM službě.

```
AuthPAMService jméno_služby
```

Určuje jméno služby, proti které se bude autentizace provádět.

[4] `https://github.com/adelton/mod_authnz_pam`

6.2.2 Vnitřní struktura

Jako první dochází opět k registraci modulu pomocí hákovacích funkcí. Konkrétní funkce závisí na používané verzi serveru Apache a pro analýzu řešení nejsou důležité - nginx používá odlišné mechanismy a fáze. Nejdůležitější informací je, že se registrují do fáze zpracování, která odpovídá přístupové fázi serveru nginx.

Při samotném obsloužení požadavku je nejprve načtena konfigurace modulu a poskytnuté parametry jsou zvalidovány a zpracovány pro další použití. Dále je načtena hlavička požadavku, ze které je získáno jméno uživatele, který má být autentizován a/nebo autorizován. Nakonec je volána funkce pam_authneticate_with_login_password(), která obstarává veškeré podstatné funkce modulu - komunikaci s PAM a volání PAM API funkcí.

Funkce `pam_authneticate_with_login_password()` přijímá jako parametry strukturu požadavku typu, jméno služby, uživatelské jméno a heslo a příznaky určující, jaké kroky budou provedeny - autentizace, autorizace nebo obojí. Heslo je použito pouze, pokud je modul použit jako autentizační mechanismus Basic autentizace. V takovém případě je pak konkrétní registrační funkcí zajištěno, že bude mít modul heslo k dispozici.

Na začátku funkce `pam_authneticate_with_login_password()` jsou nadeklarovány potřebné struktury a proměnné, které budou potřeba pro zahájení sezení s PAM. Je vytvořen ukazatel na vnitřní strukturu PAM typu `pam_handle_t` a sestavena konverzační struktura obsahující ukazatel na funkci obstarávající komunikaci (např. alokace paměti pro odpověď) a heslo.

Následně je volána funkce `pam_start()`. Té je předáno jméno služby, jméno uživatele, a adresy struktur.

V případě, že dojde k úspěšnému navázání spojení, se na základě nastavených příznaků volá autentizační funkce `pam_authenticate()` a autorizační funkce `pam_acct_mgmt()`. Zmíněné funkce byly popsány v kapitole 4. Tím v podstatě celá komunikace končí a sezení je ukončeno voláním funkce `pam_end()`.

Kapitola 7

Návrh řešení a implementace modulů

V této kapitole bude rozebrána funkcionalita a konfigurační možnosti již existující modulů, které by bylo možné použít při integraci služeb běžících na serveru nginx do FreeIPA prostředí. V případě, že žádné řešení neexistovalo nebo nevyhovovalo, je popsán postup implementace vlastních modulů.

7.1 ngx_http_auth_spnego_module

Jako alternativu pro modul mod_auth_kerb se mi podařilo dohledat modul ngx_http_auth_spnego_module[1]. Tento modul podporuje autentizaci protokolem Kerberos verze 5 za použití mechanismů SPNEGO a GSSAPI. Pro zajištění správné funkčnosti je zapotřebí mít nainstalované balíky obsahující knihovny Kerberos a GSSAPI.

Modul je šiřitelný za podmínek dvouklauzulové BSD licence.

7.1.1 Konfigurace

Pro zjištění možností konfigurace a tedy i dostupných funkcí modulu se senámíme s konfiguračními direktivami, které modul poskytuje. V rámci tohoto dokumentu nám postačí představit pouze základní direktivy pro nastavení autentizace přes Kerberos verze 5. Informace o dalších direktivách je možné nalézt na stránkách projektu (viz poznámka pod čarou).

`auth_gss on|off`

Direktiva auth_gss zapíná a vypíná použití modulu. To je v prostředí serveru nginx, kde je konfigurace děděna nižšími úrovněmi, velmi praktické.

`auth_gss_keytab cesta/ke/keytab`

Tato direktiva je ekvivalentem direktivy `Krb5Keytab` používané modulem mod_auth_kerb. Jako parametr se udává absolutní cesta ke keytab souboru.

[1] `https://github.com/stnoonan/spnego-http-auth-nginx-module`

`auth_gss_realm název_oblasti`

Odpovídá direktivě `KrbAuthRealms` z Kerberos modulu pro server Apache. Narozdíl od ní ale přijímá pouze jeden název oblasti, pro kterou se má aplikovat.

`auth_gss_service_name jméno_služby`

Ekvivalent direktivy `KrbServiceName`; jako parametr přijímá jméno služby, proti které se bude autentizovat.

`auth_gss_authorized_principal jméno_uživatele`

Tato direktiva zajišťuje stejné chování jako Apache direktiva `Require`. V konfiguračním souboru je jí možno použít několikrát a definovat tak uživatele, kteří mají práva přistupovat ke službě.

`auth_gss_allow_basic_fallback on|off`

Definuje stejné chování jako direktiva `KrbMethodK5Passwd` z Kerberos modulu pro Apache server; tedy v případě selhání Kerberos autentizace je použita záloha v podobě Basic autentizace. Stejně jako `KrbMethodK5Passwd` ji není doporučeno používat bez SSL.

Porovnáním těchto konfiguračních direktiv s direktivami modulu mod_auth_kerb je vidět, že tento modul poskytuje naprosto dostatečné možnosti k tomu, aby mohl být na serveru nginx nasazen jako alternativa k Apache modulu mod_auth_kerb.

7.2 ngx_http_authnz_pam_module

Při hledání alternativy pro Apache modul mod_authnz_pam jsem narazila na modul ngx_http_auth_pam_module[2]. Ten ale nevyhovoval použití pro účely použití.

Ngx_http_auth_pam_module poskytuje pouze Basic autentizační rozhraní pro získání uživatelských údajů, které následně autentizoval pomocí PAM. Jak už bylo ale zmíněno dříve, použití Basic autentizace vyžaduje minimálně kombinaci s SSL modulem, aby se uživatelovy údaje neposílaly otevřeně sítí. Konfigurační direktivy tohoto modulu umožňovaly pouze zapnutí/vypnutí modulu a předání jména služby.

Dle vzorové konfigurace PAM modulu pro Apache[3] potřebujeme, aby modul poznal, že už byl uživatel autentizován jiným mechanismem (např. Kerberos), a provedl pouze autorizaci.

V případě, že autentizace provedena nebyla se teprve použije záloha ve formě Basic autentizace a autorizace prostřednictvím PAM. Z těchto důvodů jsem se rozhodla tento modul implementovat zcela od začátku.

[2] `https://github.com/stogh/ngx_http_auth_pam_module`
[3] `https://github.com/adelton/mod_authnz_pam`

7.2.1 Konfigurace

`authnz_pam on|off`

Direktiva authnz_pam zapíná a vypíná použití modulu ngx_http_authnz_pam_module. Jak už bylo vysvětleno v předchozí podkapitole, umožnění vypnutí modulu z jednoho místa, je kvůli dědičnosti konfigurace, velmi užitečné.

`authnz_pam_name oblast`

Tato direktiva je obdobou Apache direktivy `AuthName`. Slouží pro naplnění položky WWW-Authenticate v hlavičce odpovědi. Ta je dále použita prohlížečem pro vyvolání dialogového okna Basic autentizace. Výchozí hodnota direktivy je nastavena na "PAM realm".

`authnz_pam_service jméno_služby`

Odpovídá direktivě `AuthPAMService` z PAM modulu pro server Apache. Definuje jméno služby, proti které se bude autentizace/autorizace provádět. Výchozí jméno služby je nastaveno na "nginx".

`authnz_pam_basic_fallback on|off`

Umožňuje zapnutí a vypnutí Basic autentizační zálohy v případě neúspěchu předchozího autentizačního mechanismu. Výchozí hodnota je vypnuto.

`authnz_pam_expired_redirect_url adresa_pro_přesměrování`

Definuje URL adresu stránky pro zadání nových přihlašovacích údajů. Uživatel bude na tuto adresu přesměrován po zjištění, že platnost jeho hesla vypršela.

7.2.2 Implementace

Nejprve bylo potřeba definovat a inicializovat všechny struktury vyžadované serverem nginx pro kompilaci modulu.

Byla nadefinována nová struktura pro ukládání konfigurace z kontextu location (`ngx_http_authnz_pam_loc_conf_t`). Dále bylo naplněno pole struktur typu `ngx_command_t`, které definuje direktivy obsluhované modulem.

Do struktury `ngx_http_authnz_pam_module_ctx` typu `ngx_http_module_t` byly přidány ukazatele na funkce, které se volají v různých fázích načítání. Nejdůležitější z těchto funkcí je `ngx_http_authnz_pam_init()`.
Ta registruje obslužnou rutinu `ngx_http_authnz_pam_handler()`, tím že vloží její ukazatel do pole ukazatelů přístupové fáze (NGX_HTTP_ACCESS_PHASE).

Původní snaha byla registrovat modul do popřístupové fáze, aby byl modul vyňat z kontroly direktivou `satify`. K tomu by ale bylo zapotřebí přepsat část kódu jádra worker procesu, která obstarává procházení polí rutin jednotlivých fází a přidat možnost procházet i popřístupové pole. A nebo zavést novou "popopřístupovou"fázi. Jako další možnost se nabízelo přidat modul do fáze generování obsahu, ale v této fázi už mohou jiné moduly předpokládat, že je uživatel autentizovaný a narušilo by to logiku procesu zpracování požadavku.

V poslední řadě bylo potřeba naplnit daty strukturu modulu `ngx_http_authnz_pam_module`. Do této struktury se ukládají ukazatele na předchozí jmenované konfigurační struktury a pole.

Kromě nezbytných funkcí, které slouží pro registraci modulu do fáze zpracování a zpracování načtené konfigurace, byly implementovány tři hlavní vlastní funkce. Ty poskytují funkčnost spojenou s PAM a chováním modulu jako takového.

Jako první se volá funkce `ngx_http_authnz_pam_handler()`. Tato funkce přijíma jako parametr ukazatel na strukturu obsahující požadavek. Na začátku funkce je načten konfigurační kontext modulu a provedena validace zadaných parametrů. Poté se kontroluje obsah hlavičky, konkrétně přítomnost uživatelského jména.

Pokud tato položka nebyla vyplněna a zároveň byla zapnuta Basic autentizace direktivou `authnz_pam_basic_fallback`, zavolá se funkce `ngx_http_authnz_pam_return_www_auth()`, které se předá ukazatel na strukturu požadavku a hodnota předaná direktivou `authnz_pam_name`.

Funkce `ngx_http_authnz_pam_return_www_auth()` poté vyplní hlavičku odpovědi dle doporučení RFC 2617[4].

V případě, že nebyla Basic autentizace zapnuta vrací modul návratový kód odpovídající HTTP stavu 401 - Unauthorized.

Jesliže je uživatelské jméno v hlavičce požadavku vyplněno, znamená to, že předchozí autentizační modul (Kerberos nebo vyvolaná Basic autentizace), byl úspěšný a uživatele autentizoval (při neúspěšné autentizaci by byla položka v hlavičce prázdná). Pak se přistoupí k získání uživatelského jména (a případně i hesla) z hlavičky a zavolání funkce `ngx_http_authnz_pam_authenticate()`.

Tato funkce jako parametry přijímá ukazatel na strukturu požadavku, příznak určující kroky (autentizace, autorizace nebo obojí), ukazatel na lokální konfiguraci modulu, uživatelské jméno a heslo. Jsou inicializovány proměnné potřebné ke komunikaci s PAM službou. Po úspěšném zahájení sezení zavoláním funkce `pam_start()` je zkontrolován příznak nastavující kroky procesu a podle jeho hodnoty dojde k volání funkce `pam_authenticate()`. Podle návratové hodnoty je pak umodul ukončen s kódem odpovídajícím 401 - Unauthorized, nebo je zavolána i autorizační funkce `pam_acct_mgmt()`. Pokud selže autorizace vrací modul kód ekvivalentní HTTP stavu 403 - Forbidden.

Změnou oproti dříve uvedeným PAM modulům je, že se návratová hodnota funkce `pam_acct_mgmt()` je testována i na rovnost s hodnotou `PAM_NEW_AUTHTOK_REQD`. Tato návratová hodnota znamená, že platnost uživatelského hesla vypršela. To nastává např. při změně/resetování hesla administrátorem systému. V takovém případě funkce vyplní funkce položku Location v hlavičce odpovědi - jako hodnota této položky je použit parametr z konfigurační direktivy `authnz_pam_expired_redirect_url`. A modul vrátí serveru nginx návratový kód odpovídající HTTP stavu 307 - Temporary Redirect.

Při úspěšném průběhu je navrácen kód odpovídající HTTP stavu 200 - OK.

Aby bylo možné zkompilovat tento modul, je nutné mít nainstalovaný balík obsahující knihovny PAM. V prostředí systému Fedora jde o balík pam-devel.

[4] https://www.ietf.org/rfc/rfc2617.txt

Kapitola 8

Testování konfigurací

Všechny moduly byly nakonfigurovány pro webový server nginx verze 1.7.10. Testování jejich funkčnosti bylo provedeno za pomocí nástroje curl (verze 7.32.0).

Pro přidání modulů je zapotřebí nejprve pustit konfigurační skript serveru nginx s následujícími parametry:
`./configure --add-module=/cesta-k-modulu/nginx_http_auth_spnego_module`
`--add-module=/cesta-k-modulu/nginx_http_authnz_pam_module`
`--prefix=/absolutní-cesta-k-instalaci/nginx`
Tento příkaz je možné rozšířit ještě o parametr `--with-debug`, který zapne logování chybových hlášek modulů. Po provedení konfigurace je nutné server nginx zkompilovat použitím Makefile.

První testovaná konfigurace nastavila Kerberos modul, kterým bude uživatel autentizován a PAM modul, který uživateli poskytne autorizaci.

```
location /test {
    root html;
    index index.html index.htm;

    satisfy all;

    authnz_pam on;
    authnz_pam_name "Basic realm=PAM";
    authnz_pam_service "service_nginx";
    auth_gss on;
    auth_gss_keytab /etc/ngx.keytab;
    auth_gss_realm EXAMPLE.TEST;
    auth_gss_service_name HTTP/test.example.test;
    auth_gss_allow_basic_fallback off;
}
```

Jako první byl nástrojem `kinit` přihlášen uživatel, který sice je v systému, ale nemá přístup ke službě. Očekávaný výsledek byl, že autentizace Kerberos modulem proběhne v pořádku, ale autorizace přes PAM přístup zamítne. Pro účely testování byl použit následu-

jící příkaz: `curl -Lksi --negotiate -u : test.example.test/test/`

Výsledné stavy HTTP protokolu přesně odpovídají očekávanému výsledku.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Negotiate

HTTP/1.1 403 Forbidden
WWW-Authenticate: Negotiate <token>
```

Následně byl příkazem `kinit` přihlášen uživatel s povoleným přístupem ke službě. Odpověď serveru na výše zmíněný `curl` příkaz také splnila očekávání.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Negotiate

HTTP/1.1 200 OK
WWW-Authenticate: Negotiate <token>
```

Do další testované konfigurace byla přidána direktiva `authnz_pam_basic_fallback on;` a `satisfy all;` změněna na `satisfy any;`. Uživatel byl odhlášen příkazem `kdestroy` a pustil se testovací příkaz `curl -Lksi -u uzivatel:heslo test.example.test/test/`. Očekávaným výsledkem bylo selhání Kerberos modulu a použití Basic autentizace dohromady s PAM modulem.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate:"Basic realm=PAM"

HTTP/1.1 200 OK
```

V posledním testu byla ověřena funkčnost direktivy `authnz_pam_expired_redirect_url`. Uživatelovo heslo bylo změněno správcem systému. Očekávanou akcí bylo přesměrování uživatele na stránku s formulářem pro vyplnění nových údajů.
```
HTTP/1.1 307 Temporary Redirect
Location: https://ipa.example.test/ipa/ui/reset_password.html

HTTP/1.1 200 OK
```

Kapitola 9

Závěr

Tato bakalářská práce měla za úkol seznámit čtenáře s moduly serveru Apache, které jsou používáné pro autentizaci uživatelů oproti FreeIPA serveru, a navrhnout alternativy těchto modulů pro webový server nginx.

Nejdříve byly vysvětleny pojmy týkající se autentizace a autorizace uživatelů. Poté byl představen samotný projekt FreeIPA. Čtenář byl stručně seznámen s jeho architekturou a nástroji, které FreeIPA používá. V rámci této části jsme si popsali princip autentizačního protokolu Kerberos a následně i možnosti autentizačních knihovnen PAM.

V poslední části teoretického úvodu jsme se detailně seznámili s webovým serverem nginx. Popsali jsme si jeho vnitřní architekturu, způsob zpracovávání požadavků a okrajově jsme si popsali konfiguraci.

V druhé části dokumentu byly popsány poznatky z analýzy již existujících modulů pro server Apache. Byly prozkoumány jejich konfigurační direktivy a struktura kódu. Na základě získaných informací byly vyhledány nebo implementovány odpovídající moduly pro webový server nginx. Také byl popsán způsob implementace a konfigurační možnosti těchto modulů.

Práci by bylo vhodné rozšířit o implementaci alternativ modulů mod_intercept_form_submit a mod_identity_lookup.

Další návrhy pro zlepšení by bylo vhodné směřovat na nginx komunitu. Pro účely modulu ngx_http_authnz_pam_module by bylo praktické umožnit přidávání ukazatelů na obslužnou rutinu i do popřístupové fáze, nebo navrhnout novou fázi pro zpracovávání autorizace.

Literatura

[1] FreeIPA project site. `http://www.freeipa.org/`, březen 2015.

[2] Brown, A.: *The Architecture Of Open Source Applications, Volume II.* lulu.com, 2008, ISBN 978-1105571817.

[3] Dar, U.: *Nginx Module Extension.* Packt Publishing, 2013, ISBN 978-1-78216-304-6.

[4] Kukuk, T.; Morgan, A. G.: The Linux-PAM Module Writer's Guide - the official site. `http://www.linux-pam.org/Linux-PAM-html/Linux-PAM_MWG.html`, 2010-8-31.

[5] Neuman, C.: RFC 4120 - The Kerberos Network Authentication Service (V5). `http://www.ietf.org/rfc/rfc4120.txt`, červenec 2005.

[6] Rak, R.; Matyáš, V.; Říha, Z.: *Biometrie a identita člověka - ve forenzních a komerčních aplikacích.* Grada Publishing, a.s., 2008, ISBN 978-80-247-2365-5.

[7] Smith, R. E.: *Authentication: From Passwords to Public Keys.* Addison-Wesley Professional, 2001, ISBN 978-0201615999.

Příloha A

Obsah CD

- Technická zpráva ve formátu PDF v adresáři `/thesis/`
- Zdrojové kódy technické zprávy ve formátu LaTeXv adresáři `/thesis-latex/`
- Zdrojové kódy modulů `/src/`
- Vzorové konfigurační soubory v adresáři `/conf-files/`
- Soubor `README.txt` s odkazy na aktuální verze modulů a návodem na přidání modulů do webového serveru nginx.