Technická univerzita v Liberci
Hospodářská fakulta

DIPLOMOVÁ PRÁCE

2009 Petr Drbohlav

Technická univerzita v Liberci
Hospodářská fakulta

Studijní program: M 6209 Systémové inženýrství a informatika
Studijní obor: Manažerská informatika

Návrh podnikového IS se zabezpečenou datovou vrstvou

Design of corporate IS with secured data layer

DP-MI-KIN-2009-06

PETR DRBOHLAV

Vedoucí práce: Ing. Vladimíra Zádová, Ph.D., Katedra informatiky
Konzultant: Ing. Miroslav Janíček, Siemens

Počet stran: 88 Počet příloh: 3
Datum odevzdání: 5. 1. 2009

Byl(a) jsem seznámen(a) s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom(a) povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

V Liberci, 5. 1. 2009 vlastnoruční podpis

ANOTACE

Tato práce se zabývá problematikou zabezpečení dat v moderních IS/IT systémech a to jak na obecné úrovni, tak i rozpracováním studie na praktické úrovni. Praktické řešení je zde popsáno na konkrétním prostředí instituce státní správy v ČR a dále rozvedeno pro aplikaci ve standardním informačním systému obchodní společnosti. Prochází problémová místa zabezpečení dat a jejich aplikace v rámci třívrstvé architektury (klient – aplikace – databáze). Jádrem této práce je popis řešení zabezpečení přístupu k datům na úrovni klient - aplikace a poté představení možností komunikace a zabezpečení dat mezi aplikační a databázovou vrstvou. V této práci je představeno několik možných variant aplikace obecných principů do praxe včetně zhodnocení jejich kladů a záporů pro dané prostředí. Tato práce nemá primárně za cíl sloužit jako průvodce pro zabezpečení databáze, ale klade si za cíl seznámit čtenáře obecně s touto problematikou a poskytnout mu několik náhledů na možné řešení problémů, které v této oblasti nastanou bez ohledu na dané prostředí jejich aplikací.

Klíčová slova

zabezpečení dat, audit databáze, třívrstvá architektura, bezpečnost aplikací, přenos identity uživatele, virtuální privátní databáze, proxy autentizace, LDAP, atributové certifikáty

ANOTATION

This work goes into the problems of data security within the modern IS/IT systems and it does on general level as it does by developing a study on practical level. Practical application is described on particular environment of government institution in Czech Republic and than developed to application on standard information system of business company. It will go through problematic places in data security and their application within three-layer architecture (client – application – database). The core of that work is description of securing access to data on client – application level and than introducing communication possibilities between application and database layer. Several possibilities of application general principles into practice including pro-and-con analysis for particular environment are introduced in this work. This work doesn't primary intend to be the guide of securing database but it intends to introduce to reader the problematic and give him several ways of view on possible solutions of problems which will appear regardless of application environment.

Keywords

data security, database audit, three-layer architecture, applications security, user identity transfer, virtual private database, proxy authentication, LDAP, attribute certificates

OBSAH:

1 SEZNAM ZKRATEK A SYMBOLŮ

3L	Three-layer architecture	Třívrstvá architektura
AAA	Authentication Authorization Audit	Autentizace autorizace audit
AD	Active Directory	Active Directory
API	Application Programming Interface	Programové rozhraní aplikace
CA	Certification Authority	Certifikační autorita
CSM	Content Switch Manager	Správce přepínání dle obsahu
Db	Database	Databáze
DN	Distinguished Name	Rozlišovací jméno
DNS	Domain Name Server	Server doménových jmen
DoS	Denial of Service	Odmítnutí služby
DOS	Disc Operating System	Diskový operační systém
HR	Human Resources	Lidské zdroje
HSM	Hardware Security Modules	Hardwarové bezpečnostní moduly
HTML	HyperText Markup Language	Označovací jazyk pro hypertext
HTTP	HyperText Transfer Protokol	Hypertextový přenosový protokol
HTTPS	HyperText Transfer Protocol - Secure	Bezpečná verze hypertextového přenosového protokolu
HW	Hardware	Hardware
IDS	Intusion Detection Systém	Systém detekce napadení
IEC	International Electrotechnical Commision	Mezinárodní komise pro elektrotechniku
IETF	Internet Engineering Task Force	Speciální skupina pro vývoj internetových technologií
IS	Information System	Informační systém
ISO	lnternational Organization for Standardization	Mezinárodní organizace pro standardizaci
IT	Information Technologies	Informační technologie
ITAM	IBM Tivoli Access Manager	Správce přístupů IBM Tivoli
ITIM	IBM Tivoli Identity Manager	Správce identit IBM Tivoli

KDC	Key Distibution Centre	Centrum distribuce klíčů
LDAP	Lightweight Directory Access Protocol	Odlehčený protokol pro přístup k adresářovým službám
MD4	Message Digest 4	Prehled zprávy 4
MD5	Message Digest 5	Přehled zprávy 5
MIS	Management Information System	Informační systém pro řízení
MS	Microsoft	Microsoft
OAS	Oracle Advanced Security	Oracle pokročilé zabezpečení
OID	Oracle Internet Directory	Internetový adresář Oracle
OLS	Oracle Label Security	Oracle zabezpečení pomocí návěstí
OSI	Open System Interconnection	Propojení otevřených systémů
OWASP	Open Web Application Security Project	Projekt bezpečnosti otevřených webových aplikací
PKI	Public Key Infrastructure	Struktura veřejných klíčů
PL/SQL	Procedural Language / Structured Query Language	Jazyk procedur / strukturovaný dotazovací jazyk
RDN	Relative Distinguished Name	Relativní rozlišovací jméno
RLS	Row-level Security	Bezpečnost na úrovni řádků
RO	Read Only	Pouze pro čtení
RPC	Remote Procedure Call	Volání vzdálené procedury
SHA	Secure Hashing Algorithm	Bezpečnostní hešovací algoritmus
SMTP	Simple Mail Transfer Protocol	Jednoduchý protokol elektronické pošty
SQL	Structured Query Language	Strukturovaný dotazovací jazyk
SSL	Secure Sockets Layer	Bezpečná vrstva konektoru
SW	Software	Software
TCP/IP	Transmission Control Protocol / Internet Protocol	Řídící přenosový protokol / protokol Internetu
VPD	Virtual Private Database	Virtuální privátní databáze
XSS	Cross Site Scripting	Mezistránkový skript

2 SEZNAM TABULEK A OBRÁZKŮ

2.1 Seznam tabulek

2.2 Seznam obrázků

3 ÚVOD – CÍL PRÁCE

Tato práce by měla poskytnout ucelený pohled na problematiku třívrstvé architektury a její použití při řešení podnikových informačních systémů. Důležité je pochopení základních principů funkce třívrstvé architektury a její výhody oproti starší a jednodušší dvouvrstvé architektuře. Nás však zajímají základní funkční principy pouze okrajově, poněvadž hlavním cílem práce je oblast bezpečnosti dat. Systém může fungovat rychle a spolehlivě, ale pokud je zranitelný před útoky, nemá v dnešním světě šanci obstát. Platí, že největšími cennostmi v každé společnosti je know-how. To je drženo jednak v podobě znalostí zaměstnanců, ale velká část je uložena v podobě dokumentů či tabulek. Většinou je pravidlem tato data ukládat do jakéhosi centrálního úložiště společnosti, ke kterému můžou oprávněné osoby přistupovat. S těmito možnosti přístupu k cenným datům se v posledních pár letech pochopitelně našlo mnoho těch, kteří se pokusili tato data získat za účelem vlastního prospěchu či jim prostě šlo o poškození dané společnosti a vlastní zviditelnění. Jako obrana vznikla téměř nová vědní disciplína, zabývající se zabezpečením informačních technologií před útoky. Tato oblast je velice široká a po celém světě se jí zabývá statisíce lidí na plný úvazek. I díky panickému strachu z toho, že podniky využívající informačních systémů jsou odkázáni kolikrát pouze na „počítače“, stala se oblast bezpečnosti v IT velmi diskutovaným tématem a dnes již při návrhu jakéhokoliv systému nemůže být opomenuta.

Protože je obvyklé doplnit teoretické pojednání nějakým příkladem či případovou studií, v práci je pro názornost zpracováno řešení zabezpečení podnikových procesů v jedné velké společnosti státní správy. Jde o příklad řešení bezpečnosti na úrovni první a druhé vrstvy našeho třívrstvého modelu. Úkolem této práce pak bude navázat na toto řešení a rozvinout ho až na třetí vrstvu, čímž dojde k odstranění většiny slabých míst, ke kterým se v průběhu dostaneme.

Přestože můžeme při řešení zabezpečení datové vrstvy celkem s jistotou předpokládat, že to nebude první stupeň ochrany před útoky, možnosti ochrany první a druhé vrstvy jsou různorodé a často postavené na různých principech. Proto bych chtěl v rámci této práce nastínit několik možností, jak dosáhnout požadovaného stavu zabezpečení vlastních dat

v databázi včetně jejich výhod i nedostatků. Je to způsobeno tím, že neexistuje jedno univerzální řešení, jak zabezpečit data v databázi, a to z mnoha důvodů plynoucí s diverzity jednotlivých článků systému.

Na trhu s podnikovými informačními systémy funguje několik největších hráčů, jejichž produkty se dají jistým způsobem považovat za standart v dané oblasti. Většina nových technologií je v prvé řadě aplikována a propagována v produktech, které zabírají největší podíl na trhu a potřebují stále inovovat, aby udržely zájem zákazníků. Praktická aplikace řešení bude vycházet z produktu databázového serveru společnosti Oracle. To z důvodu, že samotný databázový server je velice často používán jako datové úložiště nejrůznějších dat a jeho rozšíření je prakticky globální. Navíc je firma Oracle jako taková jedním z lídrů na trhu podnikových informačních systému a vyvinula mnoho dalších produktů, které aplikují do praxe nejmodernější způsoby komunikace a zabezpečení datového úložiště (db server).

V rámci možných způsobů zabezpečení datové vrstvy práce popisuje několik speciálních utilit napojených na databázový server, které jsou vyladěny pro bezproblémovou komunikaci v rámci použité technologie. Přestože jsou tyto utility svým způsobem specifické pro dané prostředí (produkty firmy Oracle), většinou jsou přizpůsobeny k univerzálnímu použití a jsou postaveny na mezinárodních standardech a protokolech. Z tohoto důvodu lze zde navrhnutá řešení převést na jakýkoliv jiný datový zdroj, které používáme na ukládání svých dat. Pochopitelně platí, že v oblasti bezpečnosti dat jsou náklady na pokrytí všech potenciálních rizik vysoké a mohou je platit většinou pouze největší společnosti, které tak svými produkty určují standardy pro daný segment trhu.

4 ZHODNOCENÍ SOUČASNÉHO STAVU – ORGANIZACE STÁTNÍ SPRÁVY

Úkolem této práce je nalézt možnosti zabezpečení datové vrstvy tak, aby společnosti jako vlastníci dat mohli bez obav důvěřovat celému systému podnikového informačního systému. Přestože má v moderní době informace stále větší hodnotu, při vývoji aplikací v modelu třívrstvé architektury se často zapomíná na zajištění bezpečné komunikace mezi aplikačními servery (business logikou) a databází.

Analyzované prostředí představuje informační systém společnosti státní správy, kde jsou vyvíjení aplikace v současné době téměř bez jakýchkoliv bezpečnostních zásad. Hlavní výhodou je dostatečně propracovaný mechanismus firewallů na serverech společnosti, takže je interní síť kvalitně chráněna před vnějšími útoky. Nicméně dnešní nároky na bezpečnost dat jsou mnohem dále, zvláště pokud společnost pracuje s citlivými daty.

V současné době je vytvořen návrh na zabezpečení na úrovni střední (aplikační) vrstvy – projekt AAA portálu (bližší popis viz. níže). Z hlediska komplexnosti a citlivosti dat uchovávaných v datových úložištích společnosti (či v centrálním datovém úložišti) není příliš dobré se domnívat, že případné riziko úniku informací končí na aplikačních vrstvě. A nejde jen o riziko úniku dat, ale i o možnost narušení funkcionality systému.

Je běžným jevem, že aplikace přistupují do databáze pomocí tzv. technologického klienta, kdy se aplikace přihlašuje na databázový server pod jedním přihlašovacím jménem (uživatelem), který má obvykle navíc plnou kontrolu nad danou databází nebo dokonce nad celým databázovým serverem. Existují případy, kdy se zabezpečení databázového serveru příliš neřeší na úkor jednoduchosti konfigurace aplikace a její funkčnosti, a aplikace přistupuje k databázi např. pod administrátorským účtem „sa“ v SQL Serveru 2000 – tento stav by měl být pro ostrý provoz nepřípustný bez diskuze. Tento technologický klient má přihlašovací údaje pro přístup k databázi (tedy tzv. autentizační

údaje) zapsány v nekryptované podobě přímo v konfiguračním souboru aplikace – většinou „web.config“.

Přístupová práva tohoto „uživatele“ jsou nastavena na maximum logicky z toho důvodu, aby mohla aplikace zohlednit požadavky různých autentizovaných uživatelů. Tím pádem dochází k situaci, kdy uživatel aplikace, který může data z databáze pouze číst, je pro komunikaci aplikační vrstvy se samotným databázovým serverem automaticky na stejné úrovni jako uživatel, který má právo plného přístupu k datům. Jediné rozlišení by tak prováděla samotná business logika na aplikační vrstvě, tedy aplikace samotná. K tomuto nejsou aplikace primárně určené a tento postup obsahuje několik vážných bezpečnostních rizik. Bezpečnost dat v databázi je v tomto případě řešena předpokladem fyzické (přesněji technické) nedostupnosti databázové vrstvy, která je „schovaná“ za aplikační vrstvou. Pro běžného uživatele se jeví přístup do databázové vrstvy jako nedostupný.

S tímto počítá i současné řešení projektu AAA portálu, který předpokládá, že se jedná o bezpečné prostředí, jež je zabezpečeno jinými prostředky (organizační opatření, nemožnost obejít filtrování síťových vrstev ap.). AAA portál tedy svými prostředky zabezpečuje komunikaci klient – aplikace.

Využití přístupu k datům přes technologického klienta s sebou přináší i další omezení, které je v dnešní době již těžko omluvitelné a v prostředí pracujícím s citlivými informacemi obzvlášť. Jde o možnost auditu operací prováděných na databázové vrstvě. Audit u moderních databázových serveru je již tak propracovaný, že je mnohdy schopen zajistit velice užitečné informace při případném úniku informací či jejich zneužití. Dá se také využít k vyhledání odpovědných osob při výskytu nežádoucích změn v databázi včetně její struktury. To jistě může i samo o sobě posloužit prevenci ze strany uživatelů (Pokud člověk ví, že je možnost zpětného dohledání jeho konkrétních kroků, bude „našlapovat“ opatrněji, což může zabránit i nechtěnému poničení dat či struktury databáze).

Z pohledu databázového serveru se pochopitelně dá provádět audit i za stavu přístupu aplikace do databáze přes technologického klienta. To však samo o sobě postrádá smysl

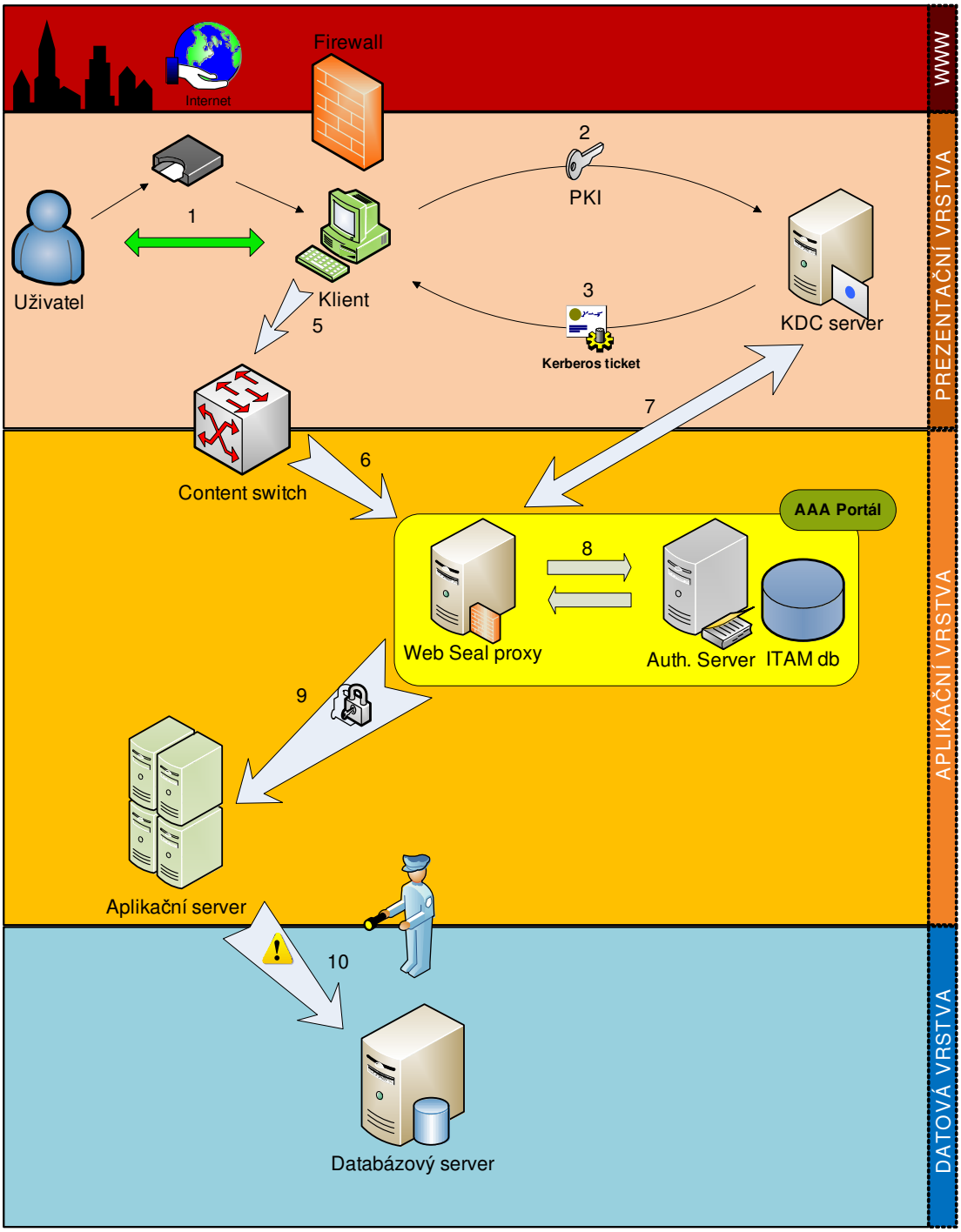
v okamžiku, kdy si uvědomíme fakt, že pod zaznamenanými informacemi o provedených změnách je „podepsán“ jediný uživatel – aplikace, tj. technologický klient aplikace pro přístup do databáze.

Výše zmíněný problém se prakticky projeví v koncepci AAA portálu tak, že po složitém procesu autentizace a autorizace uživatele je pro komunikaci s databází použit univerzální uživatel, který zastupuje všechny uživatele, mající právo pracovat s aplikací a přistupovat přes ní do databáze. Co v této databázi provádí je pouze jejich věc, protože pro tuto operaci jsou z hlediska databáze anonymními uživateli, které nelze od sebe rozeznat. V tomto pojetí pak dochází k jevu, kdy celý princip třívrstvé architektury, na kterém je založena bezpečnost dat (o ty jde především) a celý systém AAA portálu, ustupuje do pozadí na úkor pouze dvouvrstvého rozdělení, protože dochází ke **splynutí aplikační a databázové vrstvy** z hlediska business logiky – oddělené zůstávají pouze hardwarově.

4.1 Předpoklady současného stavu:

Základním předpokladem současného stavu je tvrzení, že řešení autentizace a autorizace klienta jsou pro bezpečnost dostačující ve všech třech vrstvách. Ve skutečnosti je tato bezpečnost zajištěna pouze v prezentační a aplikační vrstvě pomocí integrovaného systému přístupu k aplikacím – AAA Portálu. Ten operuje na rozmezí aplikační a prezenční vrstvy a je považován za dostatečně robustní a komplexní, že splňuje požadavky celkového zabezpečení před útoky (a to jak zvenčí, tak i zevnitř společnosti). Základním předpokladem z pohledu této práce je tak nemožnost obejít logiku autentizace a autorizace AAA Portálu a získat přístup k databázovým serverům. Toto je ze strany společnosti zajištěno dokonalou izolací databázových serverů od okolní sítě, tedy dostatečně zabezpečená páteřní síť (toho je dosaženo pomocí technologií a postupů přesahujících rámec této práce).

4.2 Popis situace:



Obr. 1 Schéma současného stavu

Zdroj: Vlastní tvorba

1. Uživatel se přihlásí na klientský počítač pomocí PKI čipové karty.

2. Klient odešle certifikát uložený na čipové kartě k autentizaci na server Active Directory – Key Distribution Center.

3. Pro autentizaci se používá tzv. lístek relace Microsoft Kerberos. Klientská část aplikace musí být tedy umět komunikovat s AD-KDC a číst kerberové tikety. Tak dojde k úspěšnému ověření uživatele a klient může zahájit komunikaci se střední vrstvou

4. Klient se chce přihlásit k aplikaci a odesílá požadavek na konkrétní data.

5. Komunikace se střední vrstvou je prováděna skrz CSM – Content Switch Manager. CSM odstraní nežádoucí provoz mezi prezentační a aplikační vrstvou. Nastavení CSM lze měnit dle aktuálních bezpečnostních zásad na firemní síti.

6. Náš předpoklad je ten, že požadavek k přihlášení k aplikaci je oprávněný a bude přes CSM propuštěn dále na jeden z WebSeal proxy serverů, které tvoří základ komunikace AAA Portálu s okolím.

7. AAA Portál potřebuje ověřit lístek relace daného požadavku, tedy Kerberos tiket. K tomu potřebuje mít přístup k ověřovací autoritě, již zmiňovanému AD-KDC serveru.

8. WebSealy ověří správnost požadavku pomocí logiky AAA Portálu, tedy odeslání požadavku na autorizační servery AAA, které jsou napojeny na databázi rolí ITAM, odkud také získají seznam rolí pro danou aplikaci pro daného uživatele. Pokud uživatel nemá právo přistupovat k aplikaci, WebSeal komunikaci uzavírá.

9. Pokud má uživatel právo přistupovat k aplikaci, WebSeal navazuje AAA HTTP komunikaci se serverovou částí aplikace (tedy s původní „střední vrstvou“). V hlavičce HTTP předává AAA serverové části aplikace tzv. ITAM credentials, které informují serverovou část aplikace o autentizovaném uživateli, který žádá

přístup a o jeho rolích v dané aplikaci. Pro serverovou část aplikace je AAA autoritou, které důvěřuje a ke klientovi přistupuje podle informací od AAA Portálu.

10. Serverová část aplikace provede požadovanou operaci a odešle dotaz na data do databázového serveru. Zde se přihlásí jako uživatel „APLIKACE“ s plnými privilegii nad databází a bez možnosti případné kontroly při zneužití. Předpokládáme tak zachování integrity ověření uživatele z AAA Portálu, která však při samotném dotazu na data již neexistuje, protože uživatel „se ztratil“ v business logice aplikace.

4.3 Hrozby:

- Ve zdrojovém kódu aplikací jsou uloženy přímé SQL dotazy na databázi či jejich šablony a to v nezašifrované podobě.

- Možnost jednoduchého zjištění přihlašovacích údajů k databázi z konfiguračního souboru aplikace, ty jsou také uloženy v nezašifrované podobě.

- Přímý SQL dotaz na databázový server bez autentizace a autorizace AAA Portálem, tedy bez užití aplikací určených ke komunikaci s databází.

- Aplikace s daty může manipulovat dle libosti (přesněji podle naprogramovaných instrukcí), proto je možné, že díky chybě v aplikaci dojde ke spuštění procesu, který nepochází od uživatele.

- Aplikace přistupuje do datové vrstvy přes technologického klienta jako plně privilegovaný uživatel.

- Autentizovaný uživatel ověřený AAA portálem se při přístupu k databázi mění v anonymního. Nemožnost auditu samotných uživatelů.

4.4 Zabezpečení klientské a aplikační vrstvy v organizaci státní správy

AAA Portál (**A**utentizace, **A**utorizace, **A**udit) je označení pro systém zabezpečující centrální správu uživatelů a s tím související jednotné autentizace a autorizace uživatelů.

Úkoly AAA Portálu by se daly popsat takto:

- **Vytvoření centrálního přístupového bodu k aplikacím** – každý uživatel bude k aplikacím přistupovat výhradně prostřednictvím centrálního webového rozhraní, které mu poskytne takové prostředí a seznam aplikací, který odpovídá jeho aktuálním rolím a oprávněním - tzv. AAA Portálu

- **Vytvoření jednotné autentizace uživatele** v rámci celého systému, autentizace bude výhradně certifikátem uživatele podle normy X.509 uloženým na PKI čipové kartě

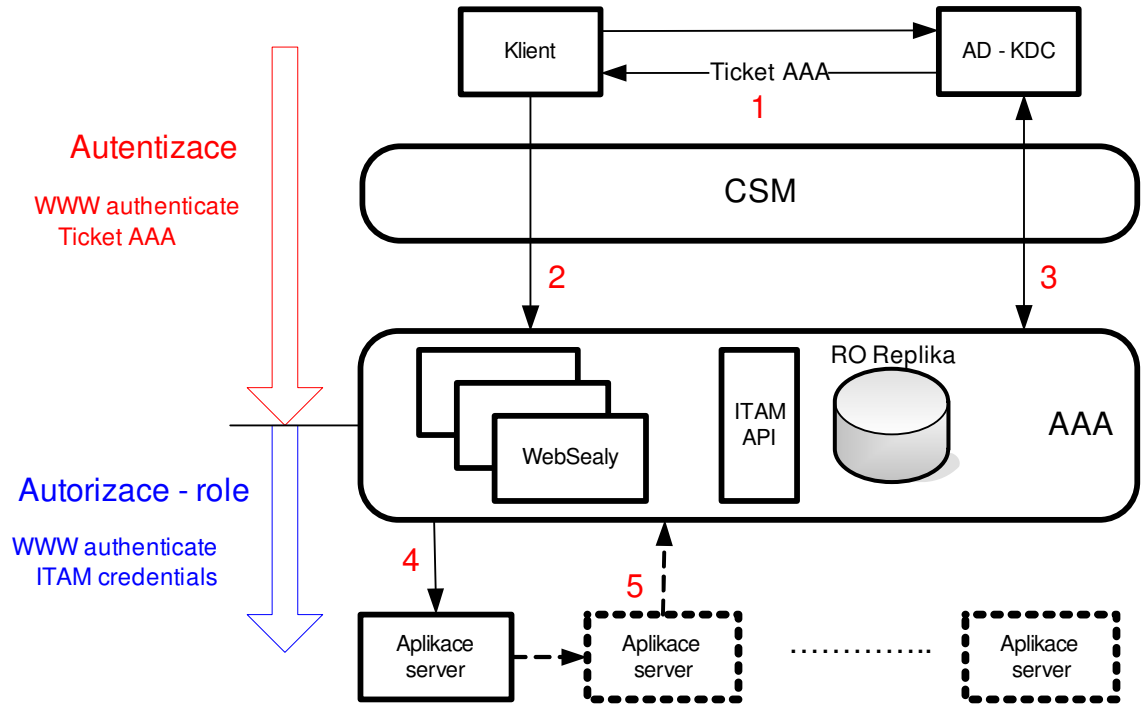
- **Centrální správa autentizace uživatelů** pomocí komponenty ITIM s propojení na personální systém Active Directory

- **Vytvoření jednotného systému autorizace uživatelů** a přidělování autorizačních oprávnění prostřednictvím komponent ITIM a ITAM

- **Vytvoření centrálního auditovacího systému**. Auditovací systém bude zaznamenávat přístup ke všem aplikacím integrovaným do AAA portálu.

Řešení by mělo umožňovat snadnou integraci i dalších aplikací, které v budoucnu budou ve společnosti implementovány, neboli řešení integrace aplikací nebude omezovat společnost ve výběru dalších systémů a aplikací ([1]).

4.4.1 Popis funkčnosti



Obr. 2 Schéma funkce AAA portálu

Zdroj: Interní dokumenty Siemens

1. Uživatel se autentizuje certifikátem na PKI čipové kartě do AD. Pro autentizaci se používá Microsoft Kerberos. Klientská část aplikace tedy musí umět komunikovat s AD-KDC a číst kerberové tikety.

2. Pokud chce klient přistupovat k aplikaci, činí tak prostřednictvím reverzních webových proxy WebSealů, které jsou součástí AAA. Pro tuto komunikaci si vyžádá od AD-KDC lístek relace – Tiket AAA, který předá AAA v hlavičce HTTPS komunikace, který AAA ověří.

3. K ověření Tiketu AAA, potřebuje mít i AAA přístup k AD-KDC.

4. WebSealy ověří správnost požadavku v ITAM, odkud také získají seznam rolí pro danou aplikaci pro daného uživatele. Pokud uživatel nemá právo přistupovat

k aplikaci, WebSeal komunikaci uzavírá. Pokud má uživatel právo přistupovat k aplikaci, WebSeal navazuje AAA HTTP komunikaci se serverovou částí aplikace (tzv. střední vrstva). V hlavičce HTTP předává AAA serverové části aplikace tzv. ITAM credentials, které informují serverovou část aplikace o autentizovaném uživateli, který žádá přístup a o jeho rolích v dané aplikaci. Pro serverovou část aplikace je AAA autoritou, které důvěřuje a ke klientovi přistupuje podle informací, které dostala od AAA.

5. Charakter některých aplikací vyžaduje také komunikaci mezi aplikacemi a následné dotazy na AAA. Taková komunikace je vyjádřena čárkovaně a její řešení je popsáno na následujícím schématu.

4.4.2 Vyhodnocování přístupových práv

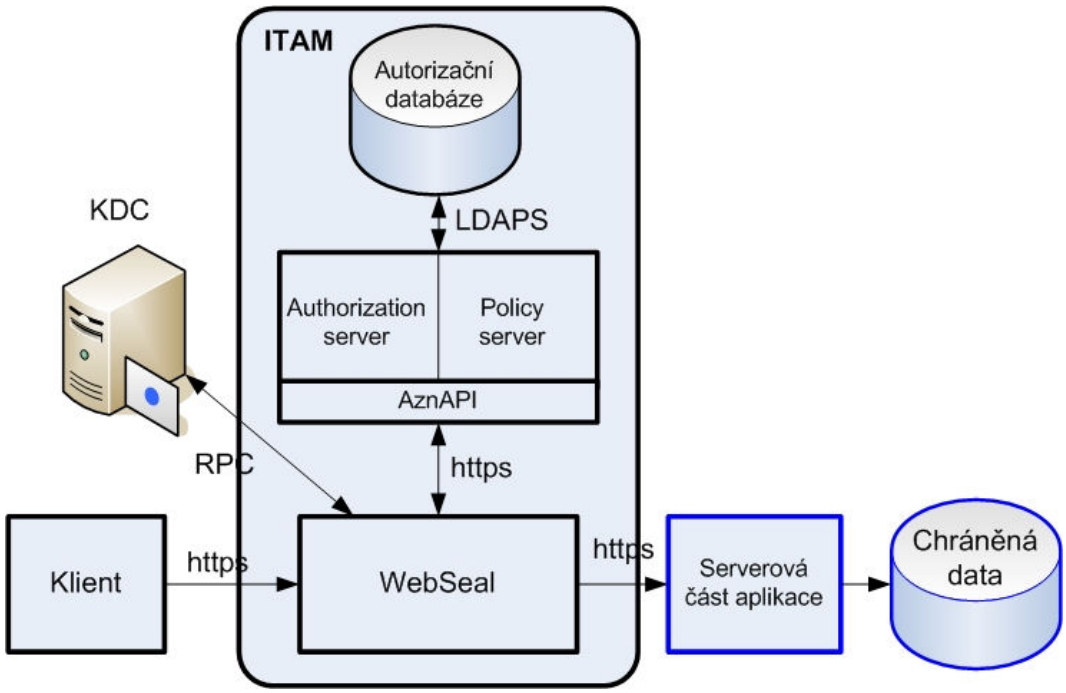
Vyhodnocováním přístupových práv se rozumí zjištění, zda daný uživatel má právo spustit požadovanou funkčnost aplikace nebo přistoupit na požadovaná data aplikace. Úloha vyhodnocování přístupových práv je rozdělena mezi AAA portál a aplikaci. AAA poskytuje aplikacím tzv. logické role, jejich faktická interpretace je na aplikacích – o tom co logické role znamenají neví AAA nic.

- AAA portál slouží pro získání seznamu logických rolí přihlášeného uživatele.

- Aplikace pak musí tyto role mapovat na jednotlivé funkčnosti, popřípadě data aplikace.

Druhý bod vyhodnocování přístupových práv je nutné řešit na úrovni aplikace, protože se jedná často o složitou a výpočetně náročnou operaci, která se musí provádět s každým voláním střední vrstvy. Zátěž generovanou tímto vyhodnocováním je nutné rozdělit mezi jednotlivé aplikace. Dalším důvodem jsou možná specifika vyhodnocování přístupových práv pro každou aplikaci.

Základní vyhodnocování přístupových práv by mělo být řešeno jednotně pomocí jedné systémové komponenty, aby se předešlo různým způsobům vyhodnocování v jednotlivých vyvíjených aplikacích. Tato systémová komponenta základního vyhodnocování přístupových práv však musí být připravena na možnost implementace aplikačně specifického vyhodnocování přístupových práv.

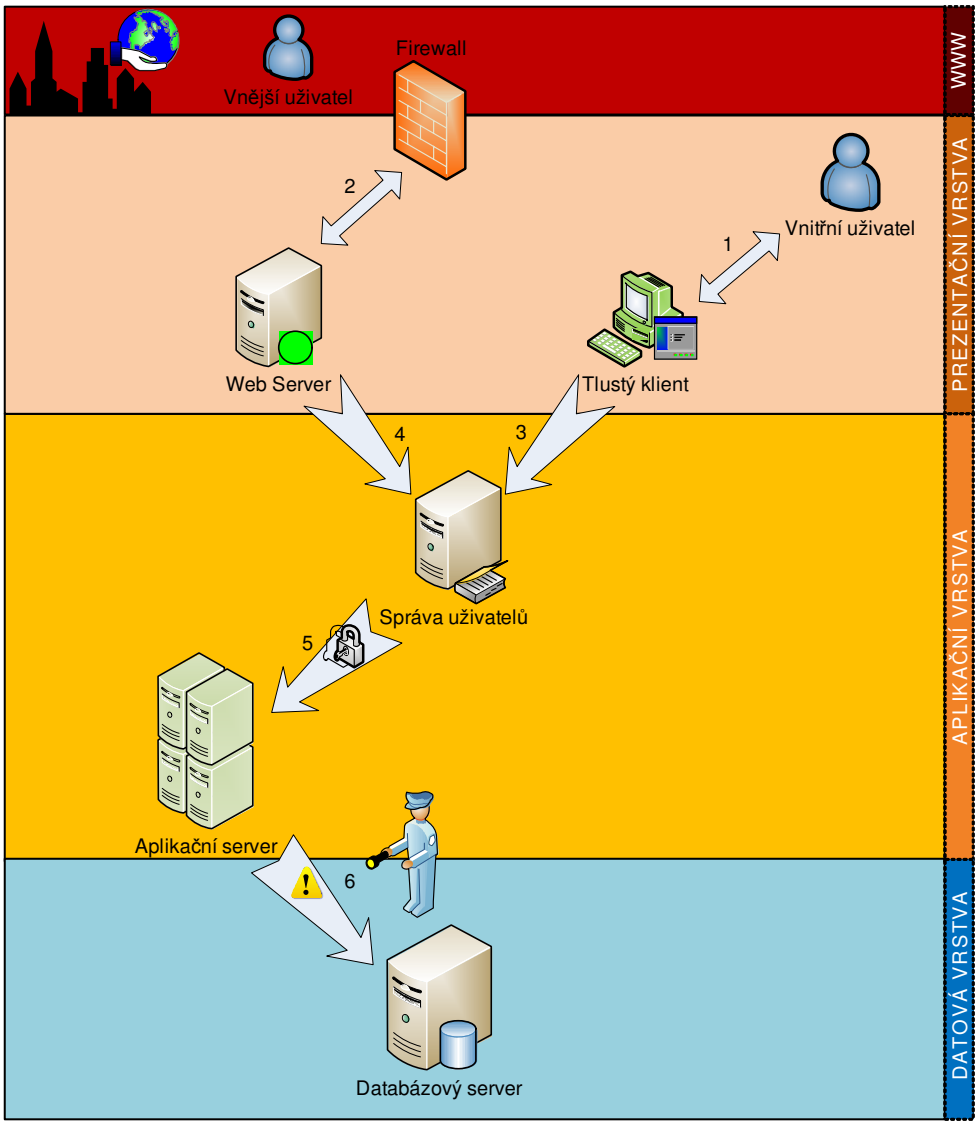


Obr. 3 ITAM poskytuje autorizační informace aplikacím

Zdroj: Interní dokumenty Siemens

5 OBECNÉ ZHODNOCENÍ SOUČASNÉHO STAVU

Problém nechráněné datové vrstvy je při použití třívrstvé architektury v praxi běžným jevem. Při zavádění informačního systému v podniku jsou náklady pro podnik v mnoha případech limitujícím faktorem, proto se autentizace a autorizace klientů provádí pouze na rozhraní mezi prezentační a aplikační vrstvou. Zjednodušeně by se dal proces identifikace uživatele v běžném podnikovém informačním systému postaveném na třívrstvé architektuře popsat takto:



Obr. 4 Obecné schéma současné situace

Zdroj: Vlastní tvorba

1. Vnitřní uživatel (zaměstnanec, administrátor) se přihlásí z lokální sítě k tlustému klientovi požadované aplikace, který je speciálně parametrizován pro konkrétní použití dle zařazení uživatele.

2. U některých společností je možné se přihlásit k aplikaci přes webové rozhraní, použitím tenkého klienta. Než však tenký klient umožní uživateli pracovat s daty, musí uživatel prokázat totožnost (uživ. jméno a heslo nebo certifikát)

3. Tlustý klient přenese identitu uživatele na centrální správu uživatelů, která funguje v rámci celé lokální sítě a přiřadí danému uživatelskému jménu potřebná oprávnění nebo zamítne identitu uživatele a tlustý klient nahlásí chybu.

4. Webový server, potažmo tenký klient aplikace běžící na tomto serveru, přenese identitu uživatele stejným způsobem jako tlustý klient, protože centrální správa uživatelů je použita pro identifikaci všech uživatelů připojených k podnikové lokální síti.

5. Pokud centrální správa uživatelů autorizuje klienta a udělí mu práva k práci s danou aplikací, požadavek se dostává na aplikační servery a aplikace tento požadavek zpracuje.

6. Jelikož aplikace předpokládá, že uživatel je již autorizován pro práci s daty, spojení mezi aplikačním serverem a databázovým serverem je považováno za bezpečné a komunikace probíhá na co nejjednodušší úrovni, většinou bez dalšího přenosu identity uživatele na databázový server, který ze svého pohledu poskytuje data pouze aplikaci, nikoliv konkrétnímu uživateli.

6 TŘÍVRSTVÁ ARCHITEKTURA V RÁMCI PODNIKOVÝCH IS/IT

6.1 Komponentové architektury

Architekturou systému lze nazývat soubor specifikací jeho částí, spojujících prvků, pravidel chování a vzájemných interakcí.

Základní myšlenkou komponentových architektur je snaha pojmout tvorbu programu obsluhujícího daný proces jako skládání jinak do značné míry nezávislých komponent. Tento koncept lze uplatnit jak na úrovni návrhu budoucí architektury, tak na úrovni implementace programu do stávajícího prostředí. Tato idea však nabývá u každé vrstvy jiné podoby, a proto je budeme dále bude popisovat odděleně ([15], s.1).

Samotná myšlenka hierarchické dekompozice je praktikována již delší dobu při tvorbě jakéhokoliv zdrojového kódu programu. Omezení však spočívá v závislosti používaných jednotek dekompozice (jako moduly či třídy) na konkrétním prostředí implementace programu. Námi chápané komponenty tyto vazby nemají, proto posouvají tuto hierarchickou dekompozici na vyšší úroveň. Pro jednoduchý příklad si stačí představit komplexnější prostředí, ve kterém probíhá velké množství procesů a každý tento proces by musel být v nejhorším případě obsluhován jedním programem. Pokud budeme mít k dispozici metodu, pomocí níž lze již vyvinutou komponentu efektivně přizpůsobit potřebám různých programů, dosáhneme velké výhody v podobě vícenásobného použití tohoto kódu. Výhody z toho plynoucí lze shrnout jako:

- Sjednocení formátu dat použitých ve všech programech a z toho plynoucí možnost jejich efektivnější spolupráce

- Zvýšení produktivity práce vývojového týmu. Opakovaně použitý kód není v prvé řadě nutno vyvíjet pokaždé od nuly, ale je možno ho převzít a pouze přizpůsobit jiným potřebám.

- Dále se díky tomuto velkému výskytu vyplatí investovat prostředky do optimalizace kódu a na ošetření případných chyb, což vede ke zvýšení rychlosti a spolehlivosti prováděných operací.

Na druhou stranu je ovšem třeba zmínit možná nebezpečí, které mohou nastat při aplikování stejné komponenty na různé problémy:

- Zvýšení některých částečných nákladů. Můžeme očekávat, že celkové náklady na vývoj klesnou, ale část nákladů může vzrůst. Zejména na straně dodavatele komponent, který musí při vývoji brát v úvahu potřebnou univerzálnost kódu a tím vyšší náročnost vývoje. I při opakovaném aplikování dané komponenty musíme stále brát v úvahu náklady na pořízení/vývoj této komponenty. Někdy se jeví jako opravdový problém tyto náklady ve společnosti uhlídat.

- Pokud nemáme jasnou představu o vyvíjeném programu, bude obtížné vybrat vhodnou komponentu

- Spolupráce mezi dodavatelem komponenty a programu nemusí vždy fungovat ku prospěchu věci. Pokud tato komunikace vázne (lze očekávat rozdíly v prioritách obou stran), lze očekávat v budoucnu problémy s údržbou jednotlivých programů.

6.2 Modely vícevrstvých architektur

Základy vícevrstvých modelů vznikly na počátku rozvoje lokálních počítačových sítí spolu z možnostmi vzájemné komunikace do té doby oddělených částí systému pomocí této sítě. Vznikl tak koncept rozdělující části programu přistupujícího k informacím z databáze do jednotlivých vrstev podle funkce. Model dvouvrstvé architektury je dnes již považován za překonaný, ale k pochopení moderní koncepce tří vrstev slouží výborně díky základnímu oddělení klienta od databáze. Dvouvrstvá architektura rozlišuje:

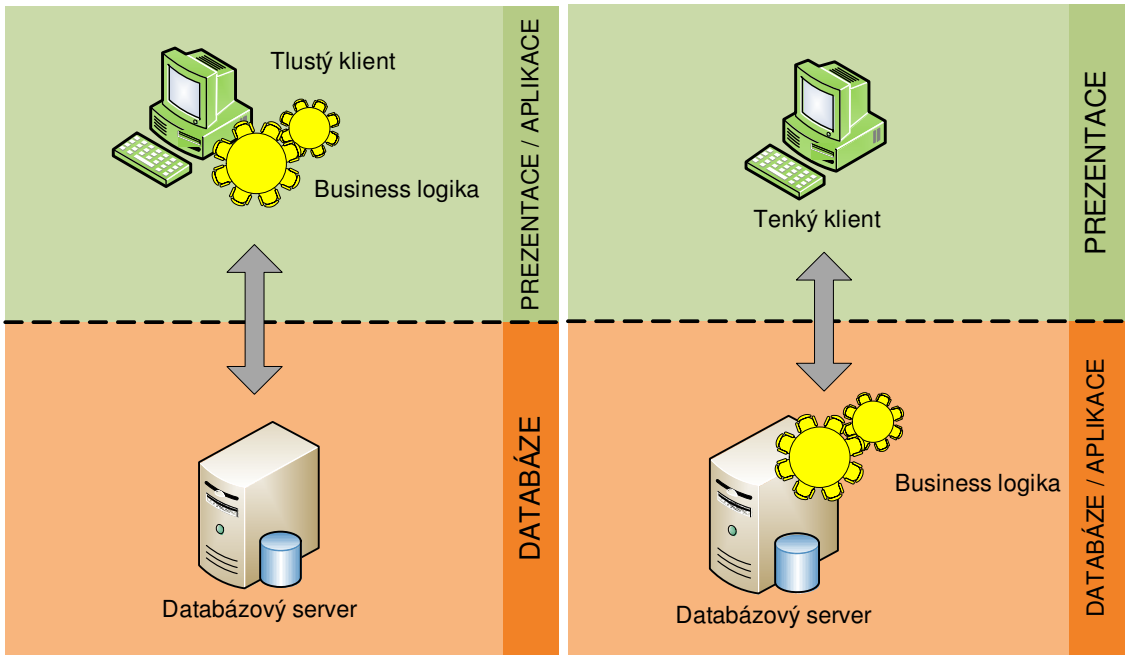
- **Prezentační (klientská) vrstva** - obsluhuje jak interakci uživatele a programu, tak i obsahuje většinu samotné business logiky aplikace. Takto klient operuje prakticky přímo nad datovým zdrojem (např. zdroj dat je reprezentován relační databází a klient pomocí jazyka SQL přímo pracuje s daty)

- **Datový zdroj -** ten bez ohledu na použitého klienta umožňuje přístup k databázi. V závislosti na prezentační vrstvě pak na této úrovni mohou běžet i některé funkce programu, ale většinu práce stejně musí odvést klientská část aplikace (tzv. tlustý klient, viz. níže)

6.2.1 Tenký / Tlustý klient

Tenký klient umožnil přesunout aplikační logiku na aplikační server a slouží tak výhradně jako „prezentační vrstva“ v takové podobě, jak by si představil laik vrstvu prezentující vstupy/výstupy. Tenké pojetí klientu dovoluje lépe zvládnout situace, kdy je potřeba v rámci údržby vyvíjeného programu spravovat mnoho klientů, u nichž je často i problém s administrací a updaty složitějších aplikací (díky vzdálené lokalitě, nedostupnosti po síti kvůli bezpečnostním pravidlům atd.). Klientskou vrstvu totiž může představovat např. webový prohlížeč se sadou appletů, který má pouze minimální nároky na údržbu. Navíc přesunutí aplikační logiky mimo počítač klienta umožnilo využití moderních nástrojů pro optimalizaci výkonu aplikačních serverů (load balancing, fail over atd.)

S tlustým pojetím klientu lze naproti tomu lépe zvládnout situace, kdy jsou hlavní funkce programu natolik složité, že by příliš zatěžovaly server. Proto je vhodné, aby tyto operace prováděly jednotlivé klientské stanice a poskytly tak serveru prostředky na samotný přístup k datům. Vrstvu serveru může pak představovat např. samotná databáze (databázový server Oracle 10g či MS SQL Server), jak jsme se s ní setkali u dvouvrstvé architektury. Na závěr je třeba podotknout, že není pravda, že by tenký klient neobsahoval žádnou business logiku, ale ve srovnání s tlustým klientem jde o zanedbatelnou část.



Obr.5 Tlustý a tenký klient

Zdroj: Vlastní tvorba

6.2.2 Třívrstvá architektura

Proč však dnes již není dvouvrstvý model příliš používán? Těmi nejdůležitějšími důvody jsou univerzálnější použití a lepší možnosti zabezpečení informací a zaručení optimálního chodu v třívrstvé architektuře. Každá vrstva poskytuje navenek určité rozhraní určené ke vzájemné komunikaci. Nenastane tedy žádný problém, pokud je rozhodnuto např. změnit poskytovatele datového úložiště. To samé platí i pro prezentační vrstvu, kde dnes již mnoho aplikací je schopno standardně pracovat s více zdroji vstupů od uživatele. Jak tedy vypadá vylepšený model třívrstvé architektury?

Model třívrstvé architektury (3L) je logickým pokračovatelem dvouvrstvé architektury. Na rozdíl od dvouvrstvé architektury, klientovi není nikdy dovoleno přímo komunikovat s datovou vrstvou, tuto operaci zásadně provádí aplikační vrstva a klient nemá možnost obejít tzv. business logiku procesu (aplikace). Použití třívrstvé architektury neznamená, že pro každou vrstvu musí být vyhrazen samostatný počítač. Velmi často se setkáváme se situací (zvláště u jednodušších procesů), kdy jsou všechny tři vrstvy provozovány na jednom počítači. Prezentační (klientská) vrstva není jen dominantou samotných klientů

(tlustý klient), ale může být umístěna také přímo na serveru – např. u všech služeb užívajících server ke generování HTML stránek (tenký klient).

Nevýhody dvouvrstvého modelu se ukázaly se vzrůstající komplexností klientských aplikací. Se složitostí aplikací vzrůstaly výkonové nároky na klientské počítače, což byl jeden faktor. Další faktory přímo vyplynuly z masivního rozšíření aplikací, softwarové firmy byly nuceny pružně reagovat na poptávku a tím pádem se snažit operativně plnit přání zákazníků. Z těchto business procesů byly definovány požadavky na budoucí aplikace. Například nutnost sdílení zdrojů, omezení datového přenosu atd., diky kterým se začalo uvažovat o architektuře, která by tyto požadavky pokryla.

Řešení se našlo v podobě přidání třetí - střední vrstvy (middle tier). Samozřejmě se nejednalo pouze o přidání vrstvy, bylo nutno specifikovat jednotlivé vrstvy resp. jejich role v architektuře. To vedlo k definici třívrstvé architektury.

Model 3L rozlišuje tyto vrstvy:

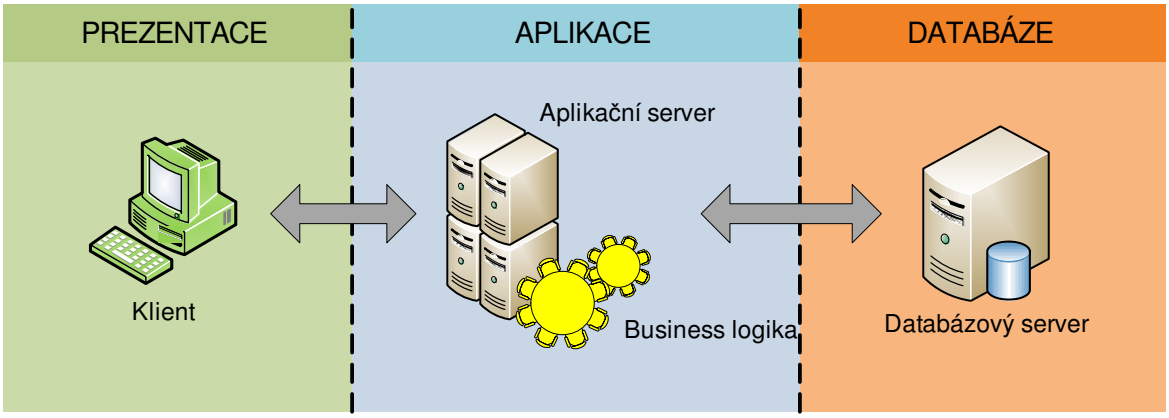
- **Prezentační vrstva** – obsahuje funkce uživatelského rozhraní. Obvykle se můžeme setkat s existencí několika prezentačních vrstev pro různé druhy zařízení, platformy a prostředí. Tato vrstva může nabývat mnoha různých podob, ale její hlavní funkcí je získat od uživatele informace potřebné k tomu, aby aplikace mohla splnit požadovaný úkol (tzv. tenký klient). Samotnou aplikační logiku bylo možno oddělit od této vrstvy a přesunout ji do střední vrstvy a tím také na výkonnější servery s rozsáhlejšími možnostmi správy a optimalizace aplikace.

- **Aplikační vrstva** – tvoří prostředníka mezi vrstvou prezentační a vrstvou datovou. Její páteří je tzv. business logika aplikace. V této vrstvě dochází ke zpracování vstupních požadavků a jejich transformaci na dotaz na datovou vrstvou na základě komponent a procesů tvořících tuto business logiku. Výborně umožňuje rozložit zatížení sítě mezi klienty a databázové servery použitím oddělených síťových okruhů.

- **Datová vrstva** – obsahuje funkce pro přístup k informacím v datovém úložišti. V dnešní době tuto funkci zajišťují sofistikovaná řešení databázových serverů předních dodavatelů SW. Její vnímání se od dvouvrstvé architektury neliší.

Nezanedbatelnou výhodou je možnost snadné implementace různých klientů pro jeden aplikační server, takže není problém jednoduše přizpůsobit klientské aplikace požadavkům různých skupin uživatelů, aniž by bylo nutné přenášet vlastní aplikační kód (narozdíl od dvouvrstvých architektur). To zároveň přináší širokou platformní nezávislost. Aplikační a databázový server může být provozován na výkonném serveru s robustním operačním systémem, zatímco klienti mohou být spouštěni na libovolné platformě. Systém s takovouto architekturou lze tedy provozovat na jediném počítači stejně dobře jako ve velmi rozsáhlé heterogenní síti. Za další související výhodu můžeme považovat i fakt, že úprava klientské aplikace většinou nemůže vnést chybu do aplikační logiky, díky čemuž uchrání společnost od možných dodatečných nákladů.

Nezávislost na platformě, velká flexibilita a snadná distributovatelnost tuto architekturu přímo předurčuje k vytváření otevřených systémů a maximálně usnadňuje systémovou integraci. Integrovat dva různé dvouvrstvé systémy využívající zcela rozdílné prostředky lze mnohem obtížněji, než vícevrstvé aplikace, u nichž je potřeba zajistit integraci pouze na úrovni aplikačních serverů.



Obr. 6 Třívrstvá architektura

Zdroj: Vlastní tvorba

Ve složitějších aplikacích je možné definovat i více než tři vrstvy. Vždy je ale od sebe oddělena prezentační vrstva od vrstvy datové. Příkladem může být např. vrstva pro kontrolu přístupových práv a zabezpečení, vrstva pro správu systémových prostředků, apod. Na výše zmíněné vrstvy se dá ale také nahlížet jako na podvrstvy aplikační vrstvy z třívrstvé architektury, stejně jako na samostatné vrstvy. Záleží pouze na úhlu pohledu. Je jasné, že v případě velmi složitých struktur může být rozdělení na více vrstev velice užitečné z hlediska přehlednosti návrhu, vývoje i implementace.

6.2.3 Logická vrstva vs. Fyzická vrstva

Závažný problém, který mate hodně vývojářů aplikací a jeho nepochopení může mít negativní dopad na výslednou aplikaci, je rozlišení mezi logickou a fyzickou vrstvou.

Logická vrstva znamená, že jednotlivé vrstvy jsou odděleny z hlediska sestavení aplikace (assembly) nebo sadami tříd aplikace, to vše ale provozováno na jednom fyzickém serveru. Naproti tomu fyzická vrstva představuje takový případ, kdy jsou jednotlivé logické vrstvy umístěny na několika serverech a obsahují v sobě navíc kód obstarávající komunikaci mezi jednotlivými fyzickými vrstvami.

To, jestli budeme používat fyzické oddělení vrstev je velmi důležité rozhodnutí, které má vliv na budoucí výkonnost systému a na druhé straně na jeho nákladnost. Je proto třeba zvážit všechny výhody a nevýhody, které nám toto rozhodnutí může přinést.

- Fyzické oddělení vrstev jednoznačně zpomalí celou aplikaci a to z důvody časové prodlevy, která je potřeba k navázání komunikace mezi oddělenými servery. Je třeba zvážit, zda tato časová ztráta vyrovná zisk z rozdělení výkonové zátěže mezi více serverů.

- Fyzické oddělení vrstev je možné provést kdykoliv a to i po uvedení aplikace do provozu. Důležité je pouze to, abychom při návrhu logických vrstev s tímto počítali a fyzické oddělení vrstev tak bylo co nejjednodušší.

- Náklady na uvedení do provozu a údržbu fyzicky oddělených aplikací jsou podstatně větší než když jsou uloženy na jednom serveru. Potřebujeme jak více samotných serverů, tak i hardware na jejich propojení a pochopitelně se zvýší složitost celého systému, což vyústí v obtížnější údržbu.

- Fyzické oddělení vrstev je vhodné tam, kde je vysoká pravděpodobnost, že např. datový zdroj bude používat více aplikací nebo jedna aplikace využívá více prezentačních vrstev. Je pak výhodné, aby všechny aplikace byly umístěny na aplikačních serverech, datové zdroje na datových serverech atd.

6.3 Aplikační vrstva – business logika

V rámci modelu třívrstvé architektury je bezpochyby ta nejdůležitější prostřední vrstva, tedy tzv. aplikační. V oblasti podnikových IS je tato vrstva často nazývána business logikou. Tento termín asi nejlépe vyjadřuje postavení této vrstvy v procesech, které zabezpečuje informační systém společnosti. Zaměstnanec společnosti vyžaduje informace ze systému, proto nejdříve dojde k přihlášení uživatele a získání informací o tom, co požaduje. Poté musí aplikace rozhodnout, zda je tento uživatel oprávněn data vidět a po autentizaci a autorizaci musí sestavit dotaz na zdroj dat (databázi) a vrácenou odpověď interpretuje a předá zpět prezentační vrstvě, která uživateli ukáže výsledek. Z tohoto zevrubného popisu je vidět, jakou úlohu plní business logika – tedy aplikační vrstva. Pojďme si tedy shrnout základní funkce, které plní:

- Vytváří reálné business objekty (např. účty, itineráře, inventáře, závazky)

- Určuje, jak se budou business objekty vzájemně chovat

- Zajišťuje cesty a metody k přístupu a aktualizaci těchto business objektů

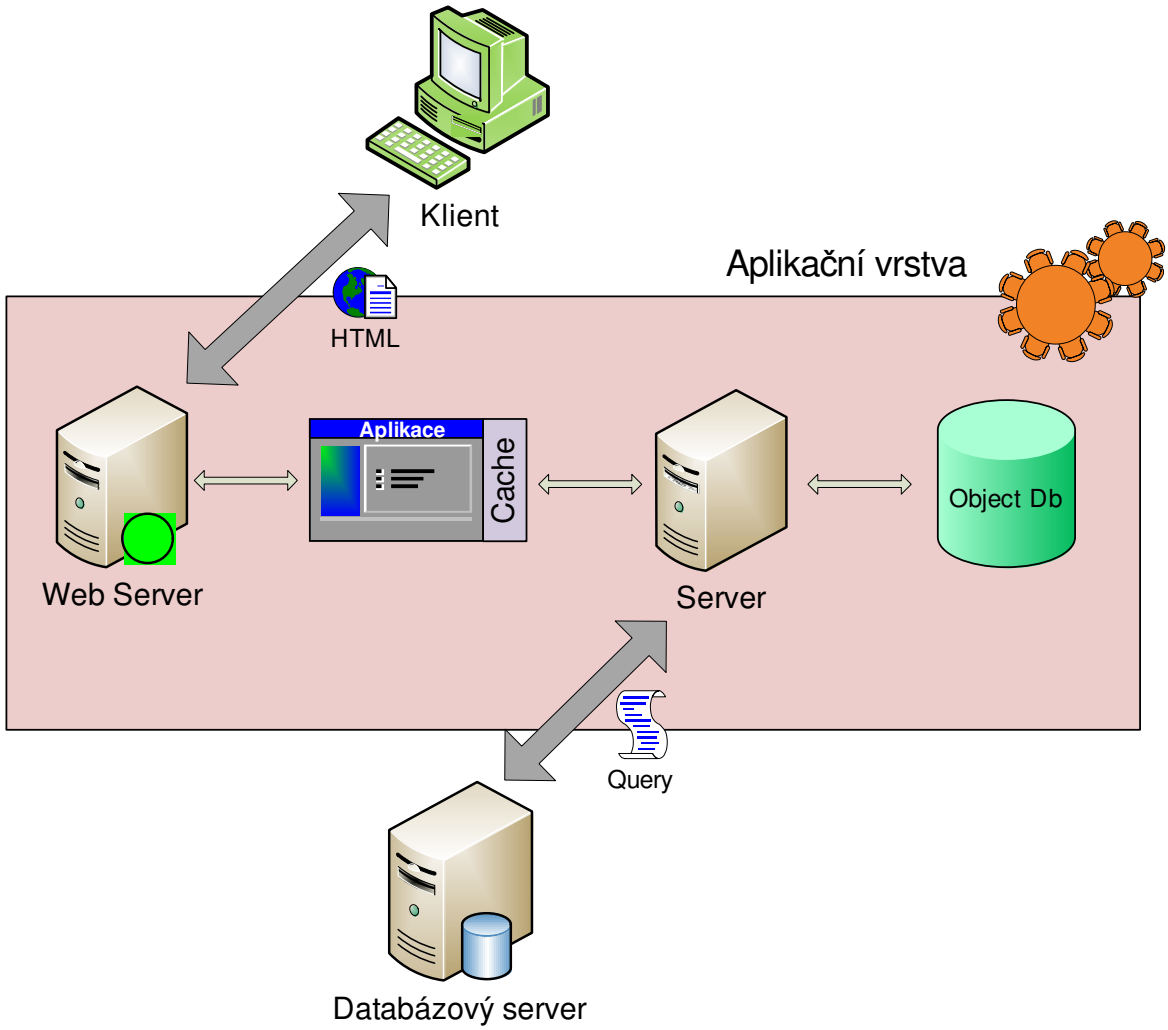
Protože chceme při návrhu procesů uvnitř informačního systému využít principů třívrstvé architektury a zde popisovaná vrstva leží uprostřed, je právě business logika zodpovědná

za jednu z největších výhod použití třívrstvé architektury. Pokud je business logika navržená správně, můžeme ji využít i v případě změn v ostatních vrstvách bez složitých úprav. Pokud změníme datové úložiště zdrojových dat, není problém upravit během krátké doby business logiku tak, aby opět plnila svoji funkci. To samé platí, pokud přejdeme k využití jiných metod prezentace dat uživateli. Právě správně navržená business logika zabezpečuje správné rozdělení aplikace do tří vrstev a tvoří tak stěžejní část celé architektury.

Pokud při tvorbě business logiky dojde k chybným řešením a nejsou rozpoznatelně odděleny jednotlivé vrstvy, dostaneme se do situace, která bude možná krátkodobě vyhovující, nicméně při následných potřebách úprav pravděpodobně narazíme na problémy, které bychom při správném návrhu neměli. Dochází tak k plýtvání zdrojů, které zvyšují náklady na vývoj a údržbu systému ([3], volně přeloženo).

Další signifikantní výhodou je to, že střední vrstva je většinou objektově orientovaná a programátoři a analytici mají zjednodušenou orientaci, protože pracují s jimi definovanými objekty a ne přímo s daty. Objektově orientovaný přístup k návrhu systémů je v dnešní době propagován hlavními tvůrci vývojového softwaru, proto jeho použití znamená implementaci nejmodernějších postupů a díky tomu máme při návrhu nejširší škálu dostupných prostředků pro komunikaci a zpracování dat. Největší výhody se dají shrnout takto: ([3], volně přeloženo)

- Jednoduché vytvoření nové komponenty
- Jednoduché testování nové komponenty
- Jednoduché nasazení nové komponenty
- Jednoduché nasazení několika verzí nové komponenty
- Jednoduchá kategorizace a dokumentace komponent



Obr. 7 Aplikační vrstva

Zdroj: Vlastní tvorba

7 ZABEZPEČENÍ APLIKACÍ V TŘÍVRSTVÉ ARCHITEKTUŘE

7.1 Bezpečnost v IT

Toto téma se v průběhu času a vývoje IT dostává stále více do popředí a prakticky u každého nového produktu z této oblasti se řeší problém, zda je tento produkt „bezpečný“ či není. Co to ale znamená bezpečnost v IT systémech? Obecně by se dalo říci, že produkt je nazýván bezpečným v té chvíli, pokud plně slouží svému účelu a zároveň neexistuje možnost útoku na funkčnost systému z řad nepovolaných osob. Za povšimnutí stojí hlavně první část tohoto souvětí, tedy že systém musí plně sloužit svému účelu, poněvadž mnoho prvků zabezpečení proti útokům je postaveno na principu omezení funkcionality daného systému, čímž jsme v mnoha případech postaveni před volbu správné rovnováhy mezi mírou bezpečnosti a rozsahem funkčnosti.

Je třeba vzít v úvahu, že tato problematika je v IT velice rozsáhlou oblastí a s vývojem stále výkonnějších strojů se rychle rozšiřují možnosti útočníků co se týká výpočetního výkonu, stejně jako s rozvojem internetu stoupá míra možné kooperace mezi útočníky. (Příkladem může být jeden z klasických hackerovských útoků na americké vládní servery, který je postaven na principu co největšího množství přístupů na server z mnoha míst, čímž by mělo dříve či později dojít k přehlcení systému a pádu serveru).

Faktem zůstává, že pokud si představíme pojem bezpečnost IT, jedná se především o bezpečnost dat jako takových. Data slouží jako zdroj informací a proto jsou velmi citlivé vůči zneužití. Bez dat, která jsou dnes většinou uložena v datových úložištích, by útočníkovi byla infiltrace do systému k ničemu.

7.2 Bezpečnost aplikací

Pro specifikaci bezpečnosti systému se vychází z bezpečnostních rizik definovaných v rámci normy ISO/IEC 17799 a projektu OWASP. Bezpečnost systému je v této kapitole popsaná z pohledu projektu OWASP, který zároveň pokrývá požadavky normy 17799 pouze z pohledu požadavků na bezpečnost systému (řešení ostatních požadavků normy 17799 není předmětem této práce) ([6], volně přeloženo).

Koncept OWASP je možné použít nejen pro webové aplikace (tedy tenký klient), ale i pro ostatní typy klientských aplikací nebo webových služeb. Důvodem je, že z pohledu dále definovaných bezpečnostních rizik jsou webové služby ekvivalentní k webovým aplikacím a budou volat služby střední vrstvy právě pomocí webových služeb.

V následujícím seznamu OWASP TOP TEN bezpečnostních rizik je popsáno, jakým způsobem by měla být tato rizika eliminována ve vyvíjených aplikacích. (Názvy jsou uváděny v angličtině, v případě, že název je snadno přeložitelný, je v závorce uveden i český překlad) ([6], volně přeloženo).

- **Unvalidated input** (Neověřená vstupní data)

  Server zpracuje data pocházející z dotazu bez jejich validace.

  Řešení: Veškerá data přicházející na aplikační server musí být validována.

- **Broken Access Control** (Narušení bezpečnosti vyhodnocování přístupových práv)

  Uživatelská práva vycházející z přiřazených rolí nejsou správně uplatněna.

  Řešení: Komponenty vyhodnocování přístupových práv v systému musí zajistit, že každý požadavek na server je autorizován.

- **Broken Authentication and Session Management** (Narušení bezpečnosti ověřování uživatelů a správa session)

  Informace o uživatelských účtech a session nejsou dostatečně zabezpečeny.

  Řešení: Zabezpečení uživatelských účtů a procesu autentizace musí být zajištěno odpovídajícími technologiemi (např. čipovými kartami, doménou, protokolem Kerberos nebo AAA portálem (viz. kap. 4.4)).

- **Cross Site Scripting (XSS) Flaws**

  Chyba ve webové aplikaci může být použita jako mechanismus pro dopravu nebezpečného kódu na lokální počítač (tenkého klienta) daného uživatele.

  Řešení: Veškeré vstupy od uživatele musí být validované a data zobrazovaná v generovaných stránkách musí být kódovaná pomocí HTML entit (řešeno ve vrstvě uživatelského rozhraní pro tenkého klienta).

- **Buffer Overflows** (přetečení zásobníku)

  K přetečení zásobníku dochází pokud základní software (operační systém, aplikační server atd.) nevaliduje vstupní data.

  Řešení: Tento problém se nevztahuje na samotné aplikace implementované technologiemi Java nebo .NET, proto je nad rámec této práce

- **Injection Flaws**

  Aplikace během přístupu na vzdálené systémy, databáze nebo lokání operační systém předávají parametry, které jsou zároveň vstupem od uživatele.

  Řešení: Validace vstupních dat. Specifický případ SQL injection je řešen kontrolou přístupových práv na data.

- **Improper Error Handling** (Nekorektní zpracování chyb)

  Chybové stavy vznikající při běžném provozu nejsou správně zpracovány. Koncovému uživateli je poskytnuto příliš mnoho informací o chybě, ze kterých je možné vyčíst citlivé informace.

  Řešení: Zpracování chyb je řízeno jednotně pomocí systémové komponenty a pravidly pro reportování chyb.

- **Insecure Storage** (Nezabezpečené nakládání s kryptografickými daty – zejména klíče)

  Webové aplikace užívají kryptografické funkce pro ochranu dat. Z důvodu problematické implementace této funkcionality a navazující složitosti kódu často dochází ke slabé ochraně.

  Řešení: Na systémy většinou nejsou kladeny požadavky, které by vyžadovaly práci s kryptografickými funkcemi. Pokud však tyto požadavky jsou, je možné pro řešení tohoto problému použít HW ochrany tzv. HSM (Hardware Security Modules). Aplikace typu tenký klient musí umožňovat ověření serverového certifikátu SSL komunikace (nejedná se však o práci s citlivými kryptografickými klíči).

- **Denial of Service** (Nedostupnost služby)

  Útočníci mohou zatížit aplikační zdroje do takové míry, že legitimní uživatelé ji nemohou dále využívat.

  Řešení: Řešení tohoto problému je velice problematické (v zásadě neexistuje ideální řešení) a je mimo rozsah této práce. DoS se standardně řeší na úrovni infrastrukturní bezpečnosti pomocí IDS (Intusion Detection Systém).

- **Insecure Configuration Management** (Správná konfigurace HW/SW bezpečnosti)

  Existence jasně definovaného standardu konfigurace serverů je kriticky důležitá pro bezpečnost poskytovaných aplikací. Servery mívají množství konfiguračních možností, které ovlivňují bezpečnost. Servery nejsou bezpečné „out of the box“, tedy ve své základní konfiguraci.

  Řešení: konfigurace serverů z pohledu bezpečnosti je předmětem infrastrukturní bezpečnosti (tedy mimo rozsah této práce).

8 AUDIT DATABÁZE

Nedílnou součástí bezpečnostní politiky každé společnosti je provádění auditu. Audit v našem případě znamená mít zdroj dat (databázi) pod dohledem a chránit ji před neoprávněnými zásahy. Důvodů pro použití auditu existuje velké množství a každá společnost by měla dle svých priorit a požadavků na bezpečnost dat zvážit, které možnosti auditu budou vyhovovat jejím potřebám. Podle zveřejněných průzkumů jsou kritéria pro audit u většiny společností jiná, unikátní.([14], s.413).

Audit může být pouze všeobecný, stejně jako může být velice podrobný, nebo cokoliv mezi tím. Můžeme zaznamenávat pokusy uživatelů o přístup do databáze (oprávněné i neoprávněné), použití systémových oprávnění, změnu struktury dat, v případě potřeby můžeme sledovat operace s jednotlivými tabulkami či řádky v tabulkách. Je zřejmé, že sledovat můžeme v databázi prakticky všechno. To však neznamená, že by to mnoho společností dělalo a už vůbec to neznamená, že bych to doporučoval jako potřebnou ochranu dat. Faktem je, že samotný audit databáze není extrémně náročný na systémové zdroje databázového serveru. Pochopitelně však práci serveru zpomaluje a záleží na konkrétní implementaci, zda je audit přítěží pro dosažení optimálního výkonu databáze. Druhou věcí je, že získaná data je potřeba interpretovat a vyhodnotit. Pokud však audit sleduje příliš proměnných, je velmi náročné se v těchto datech orientovat a smysluplně je využít. Z tohoto důvodu je nejlepší audit zaměřit na co nejméně činností, jejichž sledování nám však přinese nejvíce užitku.

Nejčastějšími důvody pro implementaci auditu dat jsou:

- Byl proveden útok na data společnosti nebo je objevena trhlina v zabezpečení – v těchto případech je obzvlášť nutné promyslet požadavky na audit, poněvadž přirozená reakce je okamžitě zapnout audit a začít sledovat vše, co se v databázi děje.

- Požadavek vyššího managementu nebo externí agentury – zde lze předpokládat, že požadavky na audit budou definovány jasněji vzhledem k potřebám těchto skupin.

- Odpovědný pracovník na základě konferencí či odborných publikací sám dojde k závěru, že je dobré audit používat a to již dříve, než dojde k viditelnému ohrožení – v případě preventivní implementace auditu je potřeba jasně určit hranici mezi požadavky na audit a zatížením systému, aby nedošlo k markantnímu zpomalení.

8.1 Možnosti auditu

Existuje několik základní oblastí, které je možné sledovat v databázi. Základní výčet je níže.

- Audit zdrojů – Sledujeme, které zdroje používá každý uživatel, jedná se tedy o velmi široký typ auditu.

- Audit objektů ve schématu – Sledují činnosti prováděné s jedním nebo více objekty ve schématu databáze. Jde naopak o úzce směrovaný a detailní typ auditu.

- Sledování příkazů – Můžeme sledovat jednotlivé SQL příkazy a to od všech uživatelů nebo pouze od vybraných podezřelých jedinců. Je to tedy typ auditu, jehož rozsah je velice široký.

- Sledování oprávnění – Sledování systémových oprávnění, které je opět možno použít na každého uživatele či na úzkou skupinu uživatelů. Jedná se o audit středního rozsahu.

Do protokolů vytvořených námi zvoleným typem auditu je možné zaznamenávat například tyto informace:

- Datum a čas
- Identifikátor terminálu
- Identifikátor relace
- Jméno nebo ID uživatele
- Použitá systémová oprávnění
- Jméno objektu ve schématu, se kterým uživatel pracuje
- Operace, kterou uživatel provedl nebo se o to pokusil
- Výsledkový kód operace

8.2 Výhody a nevýhody auditu

Hlavní výhody implementace auditu dat jsou:

- Zvýšení zabezpečení celého systému díky možnosti zpětného dohledání viníka útoku, čímž dosáhneme především preventivní obrany před zneužitím pravomocí ze strany zaměstnanců společnosti. Pokud zaměstnanci vědí, že operace s databází jsou sledovány a při porušení pravidel jsme schopni vyhledat viníka, funguje audit spolehlivě jako prevence před útoky zevnitř organizace.

- Široké možnosti sledovaní mnoha objektů a událostí a s nimi spojených parametrů, což dává odpovědnému pracovníkovi možnost dosáhnout přesně požadovanou úroveň auditu.

- Jednoduché uvedení do provozu s možností snadných průběžných úprav nastavení auditu. To je výhoda z technického hlediska, ale je třeba nastavení auditu dobře promyslet a vytvořit kvalitní plán jeho implementace.

Na druhou stranu je třeba počítat s několika nevýhodami:

- Problematika náročnosti auditu na systémové zdroje. Je jisté, že každá další auditovaná událost zpomalí výkon databázového serveru a pokud chceme provádět

audit na úrovni, která bude přesahovat možnosti aktuálního vybavení, je třeba počítat s náklady na zvýšení výkonu systému.

- Důležité je zamyslet se nad potřebami lidských zdrojů, které budou provádět vyhodnocení nasbíraných dat a vytvářet reporty o auditu. To může zabrat mnoho času a během této doby budou využity lidské zdroje, které mohou být potřeba jinde k plnění důležitějších úkolů.

Dále je třeba přesné vymezení pravomocí a odpovědnosti za monitoring auditu, což zabrání situaci, kdy je databáze plná informací z auditu a nikdo tato data nepoužívá. Tato odpovědná osoba by měla znát cíle auditu a představu managementu o způsobu prezentace výsledků. Také by měla plánovat pravidelné vytváření sestav a prezentovat je managementu. V neposlední řadě tato osoba může být pověřena administrací dat z auditu, např. sledovat velikost uložených dat a včas tato data smazat či archivovat.

Odpovědná osoba by měla také vytvořit krizový scénář, který nám může ušetřit spoustu času a starostí, pokud dojde k odhalení poškozených dat. Tento scénář či spíše strategie by měla být konzultována a schválena managementem a to dříve, než ke krizové situaci dojde. V neposlední řadě je existence takovéto strategie důležitá proto, aby zaměstnanci věděli, že máme jasný plán, jak v takových případech postupovat a odradí je to od úmyslného poškození dat.

9 PŘENOS IDENTITY A VYUŽITÍ VPD

Základním problémem, který při anonymním přístupu k databázovému serveru máme, je nemožnost identifikace útočníka při případném úniku dat či jejich poškození. Předpokládáme, že identita klienta je ověřena na aplikační vrstvě. To také předpokládá, že databázový server musí plně důvěřovat aplikačním serverům. Proto při případném útoku na aplikační server dostane útočník jako dárek neomezený přístup k datům a my navíc později ani nemusíme zjistit, odkud k úniku došlo.

Prvním krokem, který bychom měli v takovém případě udělat je získat možnost kontroly databáze pomocí auditu. Možnost auditu v našem případě znamená potřebu přenosu identifikace klienta z aplikace na databázi.

Samotný proces přenosu identity klienta na databázový server není tak složitý, pokud máme nízké nároky na bezpečnost dat při přenosu po síti mezi aplikačními servery a databází. V mnoha společnostech jsou aplikační a databázové servery umístěny ve vnitřní lokální síti a jsou odděleny od vnější sítě moderními technologiemi za statisíce až miliony korun dle velikosti podniku. Proto budeme v tomto případě předpokládat bezpečné spojení a identitu uživatele aplikace přidá k samotnému dotazu na databázi v podobě CustomerID. Z výše zmíněných důvodů nebudeme nyní řešit šifrování tohoto atributu. Pokud databázový server přijme identitu klienta, máme vynikající možnost zabezpečit přístup k jednotlivým položkám v databázi použitím komponenty VPD (Virtual Private Database, viz níže) a také jsme získali identifikátor, na jehož základě můžeme stanovit efektivní pravidla pro zavedení auditu.

9.1 VPD

Většina pomocných funkcí databázových serverů pomáhá zabezpečit data až na detailní úroveň tabulek a pohledů. Pokud však budeme chtít do důsledku zabezpečit svoje data v databázi, můžeme použít speciální metody, umožňující nám jemné odstupňování zabezpečení až na úroveň jednotlivých řádků v tabulkách a pohledech. Jedná se o součást produktu Oracle od verze 8i, VPD – Virtual Private Database. Ta slouží k řízení přístupu k datům na straně serveru a aplikačního kontextu. Ve výsledku se dostaneme až na úroveň, kdy každý individuální uživatel může vidět rozdílná data v rámci např. jediného pohledu a to na základě svých přístupových práv obsažených ve VPD. Této metodě zabezpečení se také říká Row-level Security (RLS), přičemž překlad tohoto spojení jasně vypovídá o hlavní funkci – zajištění důkladné bezpečnosti na úrovni řádků tabulky.

Využití bezpečnostních politik na tabulky a pohledy nám umožní řídit přístup k těmto datům na základě toho, který uživatel dotaz provádí. Tuto funkci sice většinou provádí business logika aplikace, ovšem přenesení tohoto úkolu přímo na databázi poskytuje jednu obrovskou výhodu. Jde o nezávislost bezpečnostní politiky na konkrétní používané aplikaci, protože zpřístupnění dat se děje až po ověření identity uživatele databázovým serverem. Ten toto ověření musí provádět vždy, než data zpřístupní. To zabrání použití jiných aplikací k přístupu k datům, než které jsou např. součástí zabezpečení vyšších vrstev v rámci organizace (např. AAA Portál). Tento fakt se dá využít i k úspoře finančních prostředků při vývoji aplikací, protože zabezpečení dat stojí vně těchto aplikací a je ho tak potřeba implementovat pouze jednou na daný databázový server . Odpadne tak složitá implementace zabezpečovacích politik do aplikací. V rámci robustního zabezpečení dat v organizaci je však třeba využít komplexního zabezpečení všech tří vrstev systému, v tomto případě lze tak brát využití VPD jako vhodný poslední stupeň ochrany dat.

V neposlední řadě je třeba zmínit další výhodu využití VPD pro zabezpečení dat v databázi. Jde také o původní využití VPD v rámci produktu databázového serveru Oracle 10i. Mnoho dnešních organizací využívá outsourcingu IT služeb či různých forem hostingu dat u jiného poskytovatele. Tito poskytovatelé jsou pak vysoce specializované společnosti využívající často osvědčený datový model pro více společností ze stejného oboru. Tyto

společnosti mohou být i konkurenty, přesto není pro poskytovatele těchto služeb problém udržovat data obou společností v jediné tabulce či vytvářet společné pohledy na tato data. Díky využití VPD lze efektivně zajistit to, že se jednotlivým uživatelům zobrazí pouze data jejich společnosti a ti tak získají pocit, že celá databáze obsahuje pouze jejich data a o existenci dat jiných společností nemají nejmenší tušení. Toho lze dosáhnout nezávisle na použitém způsobu přístupu do databáze, jak již bylo zmíněno výše.

9.2 Funkce VPD

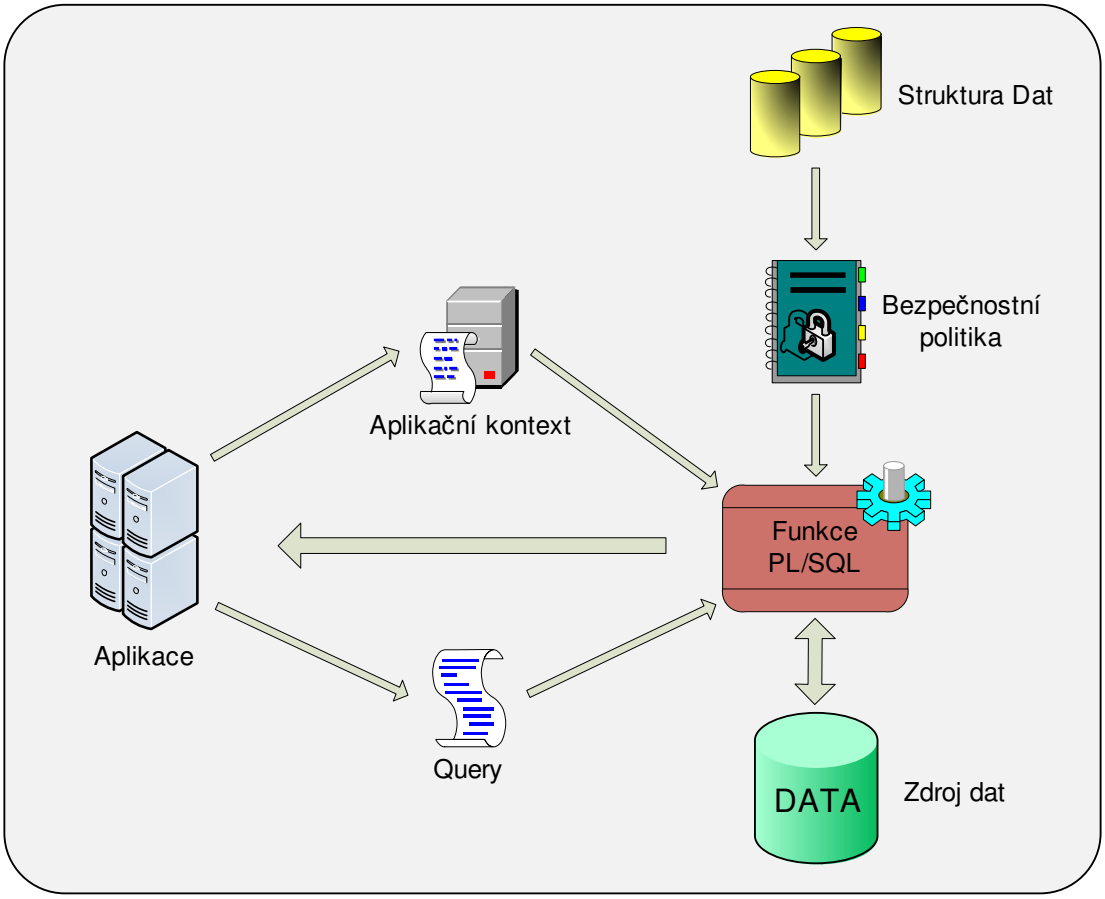
Ačkoli víme, co to je VPD a k čemu je dobrá, je užitečné zjistit, jak vlastně dosáhneme rozlišení bezpečnostních politik na jednotlivé řádky tabulek. Díky tomu, že k přístupu k datům je využíváno jazyka SQL, stačí doplnit do každého SQL dotazu přicházejícího na DB server klauzuli WHERE, za kterou doplníme jednotlivá omezení. Tato klauzule je automaticky generována na základě bezpečnostní politiky a aplikačního kontextu, který získáme při přihlášení uživatele. Takto vytvořená podmínka je pak transparentně přidána k původnímu SQL dotazu a teprve potom předána k dalšímu zpracování. Upravený dotaz je navíc optimalizován v době, kdy probíhá rozbor dotazu (tzv. parse) a je také možné ho sdílet či použít opakovaně, čímž dochází ke zvýšení výkonu databáze ([14], s. 267).

Funkci VPD si můžeme ukázat na jednoduchém přikladu:
Vedoucí jednoho obchodního oddělení ve společnosti si chce prohlédnout objednávky všech obchodníků společnosti. Vzhledem k citlivosti těchto údajů mu je v rámci bezpečnostní politiky povoleno zobrazit tyto informace pouze u objednávek uzavřených jeho oddělením. Původní dotaz napsaný uživatelem:

```
SELECT * FROM ORDERS;
```

Protože evidence všech objednávek je v jediné tabulce ORDERS, pomocí VPD dojde k přidání klauzule WHERE a funkce na základě bezpečnostní politiky vytvoří prediktát DEPARTMENT = ‘WEST‘. Výsledný dotaz tak bude vypadat takto:

SELECT * FROM ORDERS WHERE DEPARTMENT = ‘WEST‘



Obr. 8 Funkce VPD

Zdroj: Vlastní tvorba

9.3 Vytvoření VPD

Vytvoření VPD není tak jednoduché, jako konfigurace většiny ostatních komponent databázového serveru Oracle . Je to hlavně z toho důvodu, že se nejedná jen o jakousi aplikaci, která je součástí balíku Oracle, ale protože VPD je nainstalovaní přímo v databázi. Nelze tak použít standardní metody instalace a konfigurace nástrojů Oracle Universal Installer. Pro úspěšné zavedení VPD je nutné zvládnout následující posloupnost kroků: ([2], s. 294, volně přeloženo)

9.3.1 Určení databázových objektů a jejich vztahů

První fáze vytvoření VPD v našem databázovém serveru předpokládá vhodně zvolený datový model databáze, neboli strukturu databázových objektů a jejich vzájemných vztahů. Při řešení tohoto problému mohou nastat dvě situace. Buď je struktura databáze již hotova a našim úkolem bude na tuto databázi implementovat zásady přístupu k datům nebo máme možnost požadavky na bezpečnost dat zahrnout již do návrhu struktury datového modelu. Pokud se dostaneme do situace, kdy je nemožné ve stávající struktuře zvolit vyhovující model db objektů a jejich vztahů pro použití VPD, je nutné si vybrat, jestli má cenu pokoušet se vůbec využít VPD pro naše účely. V opačném případě bychom museli přizpůsobit strukturu dat požadavkům zabezpečení a to bývá u již fungujících systémů velice nákladné. Na druhou stranu toto můžeme vnímat jako upozornění na nevhodně sestavenou strukturu databáze a signál k přepracování s přihlédnutím k aktuálním požadavkům.

Poněvadž si chceme ukázat postup vytvoření VPD a její implementace na data, budeme předpokládat zjednodušenou situaci, kde vazby mezi objekty nebudou složitě členěny a orientace v nich obtížná. Ideální použití VPD je v momentě, kdy v jedné databázi ukládáme data více subjektů, kteří by neměli být schopni vidět data toho druhého. Máme tedy např. dvě obchodní společnosti, které využívají služeb jednoho poskytovatele (ASP). Obě společnosti ukládají data do tří tabulek – CUSTOMERS, ORDERS a EMPLOYEES. Prvním krokem bude stanovení primárního klíče. Každá společnost má jedinečné označení zákazníka, ovšem pouze v rámci „svých“ dat. My však musíme určit primární klíč pro celou databázi, proto musíme k číslu zákazníka přidat i identifikátor příslušné společnosti.

Struktura dat bude tak vypadat následovně:
Tabulka CUSTOMERS:

```
Create table CUSTOMERS
(COMPANY_ID varchar2(2),
CUST_ID number(9),
NAME varchar2(30),
ADDRESS_STREET varchar2(30),
ADDRESS_CITY varchar2(30),
```

```
CONTACT number(9),
EMPLOYEE_ID number(5) not null,
constraint CUSTOMER_SEC_CTX_PK primary key (COMPANY_ID, CUST_ID));
```

Poslední řádek nastavuje složený primární klíč z hodnot identifikátoru společnosti a identifikátoru zákazníka. Pouze tak můžeme zajistit jednoznačnou identifikaci zákazníka, která bude velmi důležitá pro zajištění našeho cíle.

Tabulka ORDERS:

```
create table ORDERS
(COMPANY_ID varchar2(2),
CUST_ID number (9),
ORDER_ID number (12),
ORDER_SUM number (12),
NAME varchar2(30),
ORDER_DATE date,
SHIPING_DATE date,
constraint CUSTOMER_SEC_CTX_FK foreign key (COMPANY_ID, CUST_ID)
references CUSTOMERS (COMPANY_ID, CUST_ID));
```

Vytvořili jsme cizí klíč, který nám zajistí koordinaci mezi tabulkami tak, aby hodnota, kterou vkládáme do tabulky ORDERS, existovala v tabulce CUSTOMERS.

Tabulka EMPLOYEES:

```
create table EMPLOYEES
(COMPANY_ID varchar2(2),
EMPLOYEE_ID number (5),
EMPLOYEE_USERNAME varchar(10),
constraint CUSTOMER_SEC_CTX_FK foreign key (COMPANY_ID, EMPLOYEE_ID)
references CUSTOMERS (COMPANY_ID, EMPLOYEE_ID));
```

Opět jsme museli použít cizí klíč k zajištění integrity dat v databázi mezi tabulkami CUSTOMERS a EMPLOYEES.

9.3.2 Stanovení cílů bezpečnostní politiky VPD

Nyní je třeba určit konkrétní bezpečnostní politiky, které budeme chtít nad naší databází uplatňovat. V případech složité struktury dat a rozdílných požadavků na přístup k datům nám návrh může způsobit veliké problémy. Tato fáze by se dala nazvat jako kritická. Při návrhu bezpečnostních politik pro použití ve VPD je třeba si uvědomit jednu věc – každá politika musí být definována jasně, abychom ji mohli transformovat do funkce v jazyku PL/SQL. To nás opět vrací k prvnímu bodu, tedy vhodného stanovení databázových objektů a jejich vzájemných vztahů. Bez toho by totiž vytvoření jasné bezpečnostní politiky nebylo možné. Metody používané při návrhu bezpečnostní politiky jsou různé a každému vývojáři vyhovuje jiná. Základem je však přehledné zpracování (grafické či písemné), které zajistí, abychom na něco nezapomněli a ošetřili každou možnou situaci.
Návrh našich bezpečnostních politik je jednoduchý. My chceme zajistit, aby jednotliví uživatelé viděli pouze „své“ data. Mohou tedy sledovat a upravovat údaje a objednávky zákazníků přidělených jejich číslu zaměstnance. Z toho nám vzniknou dvě bezpečnostní politiky. My si je definujeme pomocí tabulky, kde bude vše vidět jasněji:

Tab. 1 Bezpečnostní politiky

Politika	**Schéma**	**Tabulka**	**Funkce**	**Klíčové sloupce**
Uživatel vidí svá data	SECAP	CUSTOMERS ORDERS	EMPLOYEE_ID_SEC	COMPANY_ID, EMPLOYEE_ID
Uživatel vidí objednávky svých zákazníků	SECAP	ORDERS	ORDERS_SEC	COMPANY_ID, CUST_ID

Zdroj: [14], s.310

Nyní jasně vidíme, jaké sloupce budeme využívat pro aplikaci VPD. Již dříve jsme vytvořili primární klíč, který se nyní ukazuje jako velice vhodně zvolený pro uplatnění bezpečnostních politik. Může nás pouze zarazit použití sloupce EMPLOYEE_ID, ale ten je velice důležitý pro odstupňování přístupových práv na nejjemnější možnou úroveň a výborně se tak hodí k implementaci druhé bezpečnostní politiky.

9.3.3 Vytvoření aplikačního kontextu

Co je to aplikační kontext? Aplikační kontext nám umožní napsat aplikaci, která bere v potaz informace o relaci na základě přístupových práv uživatele. Aplikační kontext nabízí možnost, jak definovat, nastavit a pracovat s atributy, které může aplikace použít k jemnému řízení přístupu k datům, například pomocí námi zvolené VPD. Oracle nabízí implicitně předdefinovaný kontext USERENV (dle překladu „obálka uživatele“), ve kterém jsou obsaženy informace o aktuální relaci. Těmi nejdůležitějšími jsou jméno uživatele, identifikátor přihlášené pracovní stanice a konkrétní aplikace. V našem případě si ovšem nadefinujeme vlastní aplikační kontext, který bude však vycházet z kontextu USERNEV. K vytvoření kontextu musíme mít speciální práva Create Any context.

Než si vytvoříme kontext, musíme definovat balík a funkce, kterými tento kontext budeme nastavovat. V tabulce bezpečnostních politik je vidět, že např. pro zajištění, aby uživatel viděl pouze svá data, budeme potřebovat hodnoty atributů COMPANY_ID a EMPLOYEE_ID. Ty budeme zjišťovat z údajů právě přihlášeného uživatele, kterého identifikujeme pomocí výše zmíněného předefinovaného kontextu USERENV. Po zjištění uživatelského jména vyhledáme potřebné atributy v tabulce EMPLOYEES.

```
Create or replace package COMPANY_SEC is
        procedure GET_EMPLOYEE_ID;
End COMPANY_SEC

create or replace package body COMPANY_SEC is
        procedure GET_EMPLOYEE_ID is
        EMPLOYEE_ID_VAR number(5);
        COMPANY_ID_VAR varchar2(2);
        begin
        select EMPLOYEE_ID, COMPANY_ID into
        EMPLOYEE_ID_VAR, COMPANY_ID_VAR from EMPLOYEES
        where EMPLOYEE_USERNAME = SYS_CONTEXT (‘USERENV’,
‘SESSION_USER’);
```

```
        Dbms_session.set_context('CUSTOMER_SEC_CTX', 'EMPLOYEE_ID',
        EMPLOYEE_ID_VAR);
        Dbms_session.set_context('CUSTOMER_SEC_CTX', 'COMPANY_ID',
        COMPANY_ID_VAR);
        end GET_EMPLOYEE_ID;
end COMPANY_SEC;
```

Kromě využití kontextu USERENV jsme využili také předdefinované funkce SYS_CONTEXT. Tato funkce je velice užitečná a poskytuje přístup k mnoha dalším hodnotám, které mohou být užitečné pro tvorbu zabezpečené databáze. Výpis těch nejdůležitějších atributů je uveden v příloze A.

Po vytvoření obslužného balíku nyní vytvoříme samotný kontext, ke kterému tento balík přiřadíme:

```
create context CUSTOMER_SEC_CTX using SECAP.COMPANY_SEC;
```

Nakonec je ještě třeba, abychom zajistili nastavení našeho kontextu po přihlášení uživatele k databázi. To uděláme zavoláním funkce, kterou jsme přiřadili našemu kontextu. Funkci můžeme zavolat buď použitím samotné aplikace, která před jakoukoliv činností provede zavolání funkce, nebo použijeme trigger v databázi, kterému dáme příznak ON LOGON, tedy spuštění hned po přihlášení uživatele k databázi. Použití triggeru má neoddiskutovatelnou výhodu v tom, že je nezávislý na použité aplikaci a vyvolá se i když přistupujeme k databázi jinými prostředky.

9.3.4 Vytvoření funkce implementující politiku

Jak jsem již zmínil výše, při použití VPD implementujeme jednotlivé bezpečnostní politiky pomocí funkcí v jazyku PL/SQL a přiřazením těchto funkcí k jednotlivým tabulkám VPD. Dostáváme se tedy k finální fázi vytvoření VPD. Jak víme z popisu funkce VPD, našim cílem je vytvořit tzv. prediktát, která přidá ke každému dotazu na databázi klauzuli WHERE a za ní omezující podmínku, která umožní zobrazení pouze té sady dat, která je v souladu se zvolenou bezpečnostní politikou. Dotazem v tomto případě nemyslíme pouze

modifikaci příkazu SELECT, tedy prohlížení a výběr dat, ale máme na mysli jakékoliv operace nad danými tabulkami včetně tvorby nových záznamů, mazaní či editace.
V našem jednoduchém případě chceme, aby dotaz na databázi byl dynamicky upraven pomocí klauzule WHERE a omezující podmínkou na hodnoty sloupců COMPANY_ID a EMPLOYEE_ID. Pokud tedy chceme, aby uživatel viděl z tabulky ORDERS pouze objednávky svých zákazníků, musíme se nejdříve dotázat tabulky CUSTOMERS na čísla zákazníků daného uživatele a ty pak přidat do prediktátu nad tabulkou ORDERS. Již víme, že uživatele v tabulce CUSTOMERS identifikujeme díky COMPANY_ID a EMPLOYEE_ID zjištěných pomocí funkce SYS_CONTEXT. Uvedu zde příklad funkce napsané v PL/SQL, která řeší náš problém:

```
create or replace package body EMPLOYEE_SEC as
function EMPLOYEE_ID_SEC return varchar2 is
MY_PREDICATE varchar2(2000);
begin
        MY_PREDICATE := 'EMPLOYEE_ID =
        SYS_CONTEXT("CUSTOMER_SEC_CTX", "EMPLOYEE_ID") and
        COMPANY_ID = SYS_CONTEXT("CUSTOMER_SEC_CTX",
        "COMPANY_ID")';
return MY_PREDICATE;
end EMPLOYEE_ID_SEC;
function ORDERS_SEC return varchar2 is
MY_PREDICATE varchar2(2000);
begin
        MY_PREDICATE := 'CUST_ID in (select CUST_ID from CUSTOMERS
        where EMPLOYEE_ID = SYS_CONTEXT("CUSTOMER_SEC_CTX",
        "EMPLOYEE_ID") and COMPANY_ID =
        SYS_CONTEXT("CUSTOMER_SEC_CTX", "COMPANY_ID")';
return MY_PREDICATE;
end ORDERS_SEC;
end EMPLOYEE_SEC;
```

Předpokládejme nyní hodnoty atributů EMPLOYEE_ID = „12345“ a COMPANY_ID = „11“. Tyto funkce zajistí, že při dotazu na tabulku CUSTOMERS bude připojena klauzule WHERE ve tvaru:
where EMPLOYEE_ID = 12345 and COMPANY_ID = 11;

Při dotazu na tabulku ORDERS chceme, aby uživatel viděl pouze objednávky svých zákazníků a funkce nám proto vrátí prediktát ve tvaru:
where CUST_ID in (select CUST_ID from CUSTOMERS where EMPLOYEE_ID = 12345 and COMPANY_ID = 11);

Spojení funkcí s jednotlivými tabulkami zajistíme pomocí speciálního balíku procedur DBMS_RSL, který slouží k jednoduché administraci bezpečnostních politik. Tento balík obsahuje spoustu procedur, s nichž každá má různé parametry nastavení. Seznam jednotlivých procedur a parametrů je uveden v příloze B.

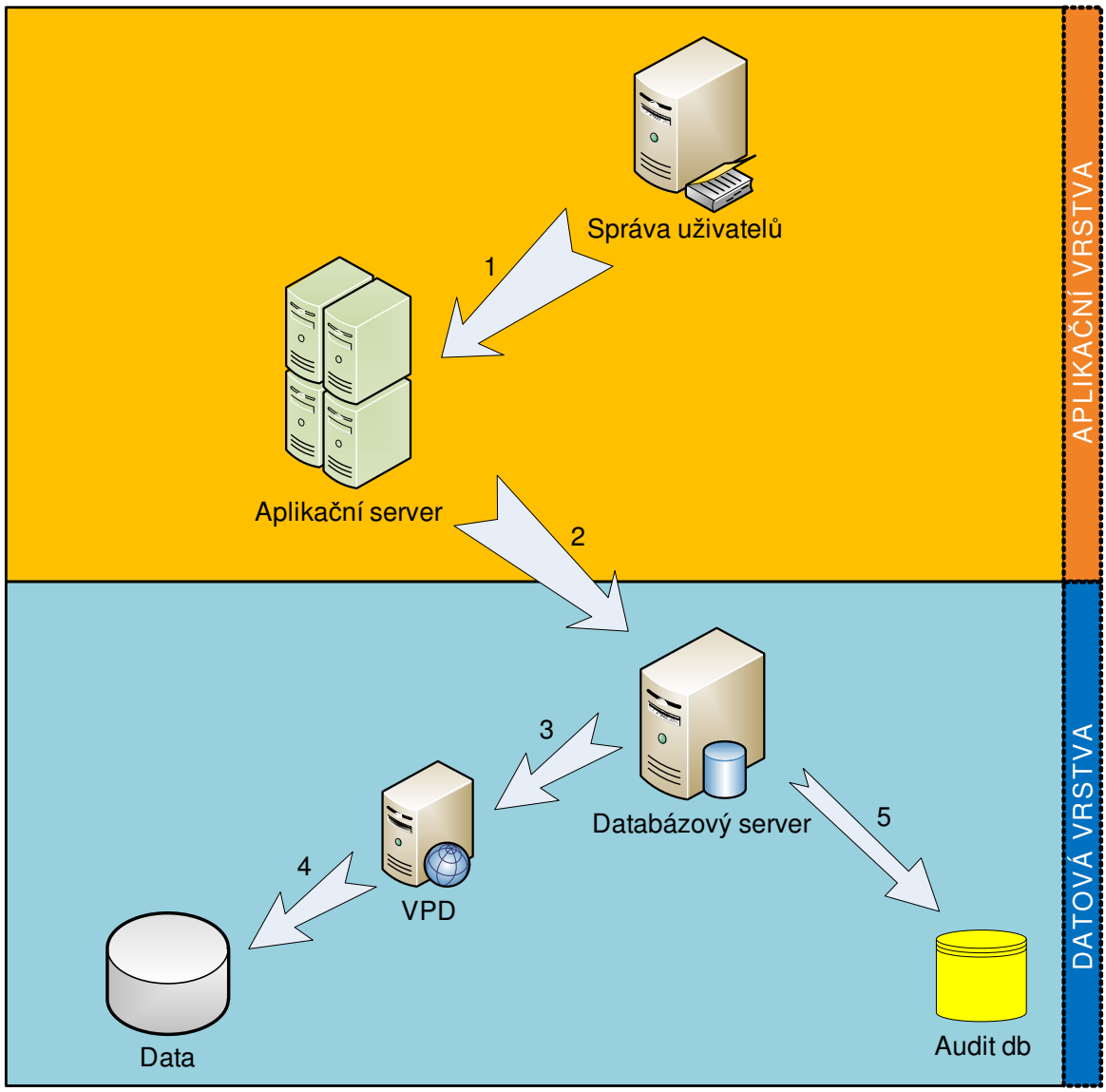
My dokončíme nastavení VPD tím, že vytvoříme politiku CUSTOMER_POLICY, která bude obsluhovat přístup k tabulkám CUSTOMERS a ORDERS. V prvním příkazu vytvoříme politiku CUSTOMER_POLICY, která nad tabulkou CUSTOMERS spustí funkci EMPLOYEE_SEC.EMPLOYEE_ID_SEC ze schématu SECAP a to pouze při provedení příkazu SELECT:

DBMS_RLS.ADD_POLICY (‘CUSTOMERS’, ‘CUSTOMER_POLICY’, ‘SECAP’,
‘EMPLOYEE_SEC.EMPLOYEE_ID_SEC’, ‘SELECT’);

Ve druhém příkazu zase vytvoříme politiku CUSTOMER_POLICY, která nad tabulkou ORDERS spustí funkci EMPLOYEE_SEC.ORDER_SEC ze schématu SECAP a to na všechny druhy dotazu, tedy SELECT, INSERT, UPDATE a DELETE ([14], s. 308 – 317, volně přepsáno):

DBMS_RLS.ADD_POLICY (‘ORDERS’, ‘CUSTOMER_POLICY’, ‘SECAP’,
‘EMPLOYEE_SEC.ORDER_SEC’);

9.4 Schéma řešení:



Obr. 9 Použití VPD pro řízení přístupu k datům

Zdroj: Vlastní tvorba

1. Centrální správa uživatelů autorizovala uživatele pro práci s aplikací a předala požadavek na data aplikačnímu serveru.

2. Aplikace (Business logika) vygeneruje dotaz na data pro databázový server. K tomuto dotazu přidá CustomerID uživatele, který data od aplikace požaduje a celý dotaz odešle na db server.

3. Databázový server předá identitu uživatele (aplikační kontext) do VPD.

4. VPD přiřadí dle dané bezpečnostní politiky uživatele do konkrétní skupiny a vytvoří prediktát, který přidá k dotazu od aplikace a teprve poté daný dotaz spustí nad zdrojem dat.

5. O prováděných operacích je uchován záznam dle nastavené úrovně auditu.

10 PROXY AUTENTIZACE

Mnoho podniků využívá aplikace postavené na třívrstvé architektuře, přesto je oddělení střední vrstvy od datové vrstvy mnohdy nejasné z hlediska přenosu identity skutečného uživatele. Často aplikace ověří uživatele na aplikační vrstvě a poté se prostě připojí k databázi jako „velký uživatel“, pochopitelně s plnými právy, Problém vznikne když potřebujeme z bezpečnostních důvodů přenést identitu uživatele až na databázi. To se dá vyřešit několika způsoby, jak je popsáno v této práci, ovšem všechny způsoby lze zastřešit využitím proxy spojení, které má obrovskou výhodu, že odděluje identity na aplikační vrstvě od datové vrstvy a tvoří jakéhosi prostředníka pro komunikaci ([5], s. 81).

Oracle poskytuje tři možnosti autentizace přes proxy: ([13], 13. kapitola, volně přeloženo)

- Střední vrstva se autentizuje s databází a klient (v tomto případě aplikace nebo aplikační uživatel) se autentizuje se střední vrstvou.

- Klient (v tomto případě databázový uživatel) není autentizován střední vrstvou, ale pomocí databázového hesla poslaného přes střední vrstvu.

- Klient (v tomto případě globální uživatel) je autentizován střední vrstvou, přes kterou pošle jednoznačné jméno (distinguished name — DN) nebo certifikát. Na základě toho získá klientské uživatelské jméno.

Uživatel se přihlašuje prostřednictvím obecného jména a hesla. Současně se ustanoví druhé, proxy spojení pomocí skutečného jména uživatele (globální uživatelské jméno nebo pomocí certifikátu). V tomto případě je identita uživatele zachována (tato metoda vyžaduje existenci účtu uživatele v databázi).

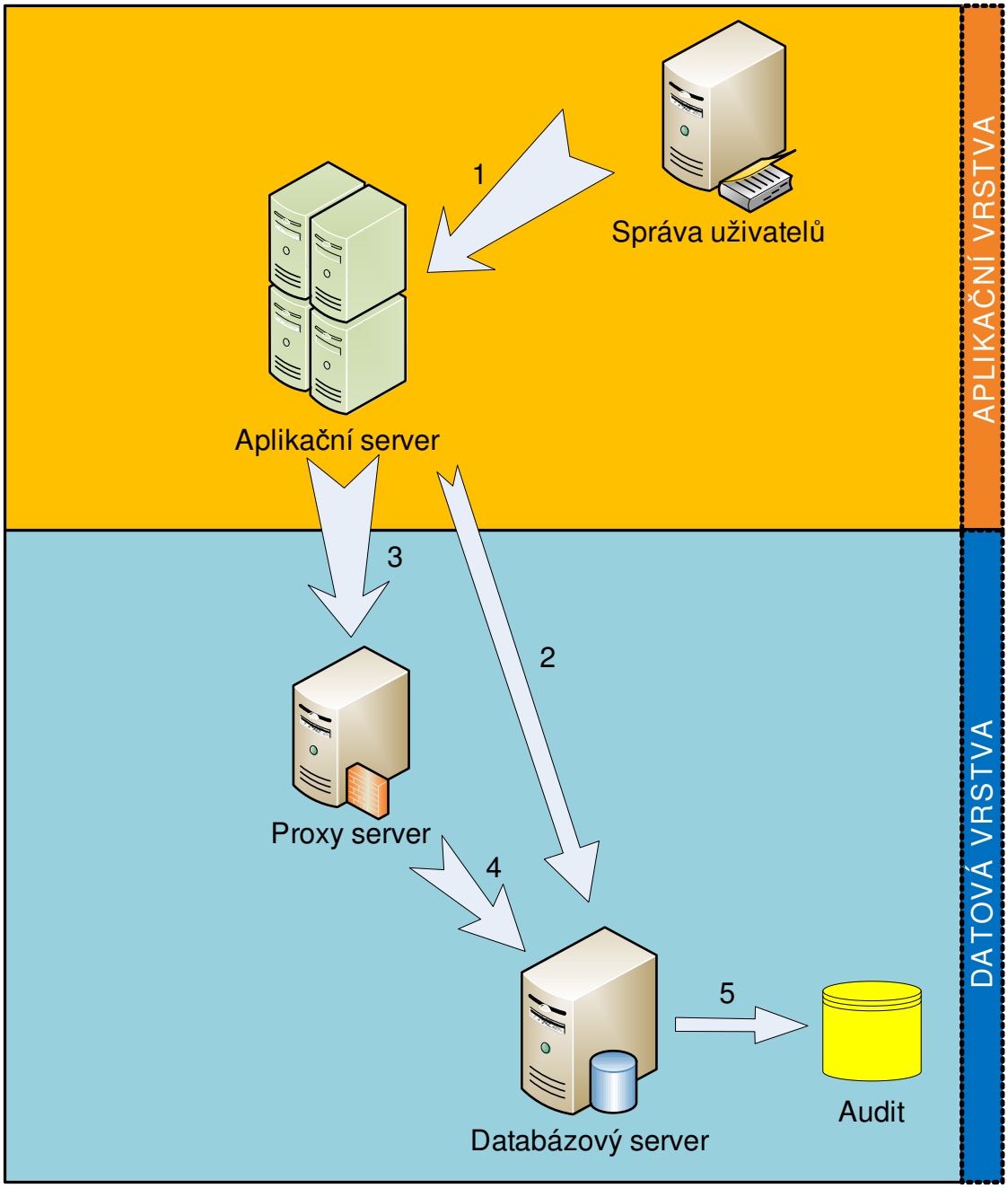
Vytvoří se databázový uživatel, tzv. proxy account, se jménem a heslem. Tento uživatel bude použit pro vytvoření connection poolu a potřebuje jediné přístupové právo: CREATE SESSION.

Pro identifikaci přístupu uživatele je použit *connection pool*, vlastní přístup do databáze probíhá pod obecným uživatelským jménem, který nemá nic společného se skutečným jménem uživatele. Na databázi přicházejí jednotlivé požadavky, které mohou používat pokaždé jiné spojení z connection poolu.

Běžný uživatel musí mít nastaveno právo použít proxy spojení přes proxy account. Databázové role při proxy připojení fungují odlišně od běžného připojení. Pro připojení přes zadaný proxy account lze určit role které budou povolené. Ostatní role, i když je má uživatel jinak přidělené, nelze při proxy připojení dodatečně povolit. Toho lze využít např. pro použití různých rolí pro různé aplikace ([13], 13. kapitola, volně přeloženo).

Konkrétní výhody, které nám proxy autentizace přináší, zmíníme v závěrečné části práce, přesto je třeba již zde připomenout základní nevýhodu proxy autentizace. Tou je poměrně složité nastavení této služby a hlavně zvýšení nároků na výkon systému a z toho vyplývající možnost kolapsu serverů a nedostupnosti služeb.

10.1 Schéma řešení:



Obr. 10 Proxy autentizace

Zdroj: Vlastní tvorba

1. Centrální správa uživatelů autorizovala uživatele pro práci s aplikací a předala požadavek na data aplikačnímu serveru.

2. Aplikace (Business logika) vygeneruje dotaz na data pro databázový server. Poté se připojí k db serveru jako obecný uživatel s právem CREATE SESSION, který otevře proxy spojení.

3. Aplikace předá skutečnou identitu klienta proxy serveru (CustomerId nebo certifikát).

4. Proxy server přeposílá skutečnou identitu uživatele na databázový server

5. Protože db server zná skutečnou identitu klienta, máme možnost dle zvolených pravidel auditovat prováděné operace.

11 POUŽITÍ LDAP

Zatímco běžná relační databáze plní funkce pro práci s velkými objemy dat a jejich častou aktualizaci, existuje i jiný druh databáze, který neukládá data do tabulek, ale v podobě adresářové struktury. Nazýváme je adresářové služby a nejpoužívanějším protokolem pro komunikaci s adresářovým serverem je LDAP (Lightweight Directory Access Protocol). Rozdílů mezi relační databází a adresářovou službou je několik a uvedeme si ty hlavní.

Relační databáze se používá pro uložení, vyhledání, vkládání a mazaní velkého objemu dat. Tato data jsou velmi rozdílné povahy a často dochází k jejich změně. Navíc relační databáze předpokládá, že tato data najde v určitých souborech na určitém stroji.

Na druhou stranu, databázi adresářové služby nejprve vytvoříte a poté do ní vložíte informace. Poté již strávíte většinu času vyhledáváním malých kousků informací, které jsou obvykle uloženy ve dvojicích (např. jméno a telefon). Tato databáze je proto orientována na jednoduché transakce zaměřené na čtení informací. Relačních informací je v adresářové databázi jen málo a většinou k jejich získání slouží i speciální klientská aplikace.

Významný rys adresářového serveru je jeho nezávislost na umístění na konkrétním místě. Aplikace vyhledávající informace je napsána tak, aby nezáleželo na tom, odkud dané informace získá. Pokud nejsou tyto informace dostupné lokálně, automaticky se aplikace dotáže na jiný server nebo sdělí klientovi, odkud může informaci získat.

Stručně řečeno, databáze adresářové služby v praxi slouží k tomu, abychom získali požadované informace v rámci různých strojů, sítí a geografických lokalit. To např. umožní klientovi sdílet vlastní uživatelské prostředí na několika strojích, aniž by si dělal starosti s jeho nastavováním na jednotlivých místech.

11.1 Funkce LDAP

Protokol LDAP je otevřený internetový standard vytvořený společností IETF (Internet Engineering Task Force), která spravuje mimo jiné i standarty protokolů HTTP, TCP/IP, DNS, SMTP atd. Samotný protokol je nástupce protokolu X.500, který byl založen na modelu OSI. LDAP využil nejpoužívanější funkce X.500 a odstranil většinu méně užitečných funkcí a vlastností. Navíc pracuje pouze na vrstvě TCP/IP modelu OSI a eliminuje náročnou režii vyšších vrstev ([14], s.320).

Pro ilustraci funkce LDAP je třeba ho rozdělit na čtyři základní modely, které ve výsledku tvoří ucelený celek adresářové služby.

11.1.1 Informační model

Tento model definuje strukturu a datové typy nezbytné k vybudování adresářového stromu LDAP. Obecně by měly být tyto položky založeny na objektech reálného světa (lidi, organizace, tiskárny atd.), ale není to žádná podmínka. Spravuje pro tyto objekty různé atributy, jimž jsou přiřazeny pravidla chování (tzv. typy). Na atributy mohou být aplikována různá omezení, jako velikost, počet a skladba parametrů atd. Tyto pravidla lze dále spojit do objektových tříd, které se chovají podobně jako schémata u relačních databází.

11.1.2 Model jmen

Většinou se záznamy tohoto modelu zobrazují jako hierarchický strom, odpovídající organizační či geografické struktuře. Každá pozice v této hierarchii má své rozlišovací jméno (DN – distinguished name). Každá komponenta tohoto jména se pak nazývá relativní rozlišovací jméno (RDN – relative distinguished name) a musí být unikátní v rámci svého rodiče (nadřazeného prvku). Princip je tedy podobný, jako adresářová struktura v operačních systémech Unix nebo DOS. U LDAP protokolu se tzv. plně kvalifikovaná cesta k objektu nazývá jmenný prostor (namespace). Ten začíná nejméně významnou komponentou a pokračuje dál až ke kořeni.

11.1.3 Funkční model

Funkční model v rámci LDAP popisuje, jak se jména používají. Rozlišujeme devět základních operací ve třech oblastech: ([14], s. 324)

- Dotazy – Slouží k prohledávání adresáře LDAP (*search* a *compare*) a získání informací. Na základě vyhledávacího filtru nalezneme požadovaný záznam včetně celé sady atributů dle naší volby. Klient navíc může určit velikost či počet vrácených záznamů stejně jako čas, který je ochoten na výsledek dotazu čekat.

- Změna dat – Volby *add, modify* a *remove* slouží k přidání, změně nebo odstranění záznamu ze struktury adresáře. V této oblasti máme také k dispozici příkaz *modify RDN*, který nám umožní měnit relativní rozlišovací jméno záznamu v rámci svého jmenného prostoru

- Ověřování – Základem zabezpečení informací v adresáři LDAP jsou funkce *bind* a *unbind*, tedy navázání a ukončení relace. Při operaci *bind* se tedy klient musí identifikovat (využití hesla není povinné, heslo se navíc udává v nezašifrované podobě), stejně jako server se identifikuje klientovi (obousměrné ověření identity). Speciální funkcí je volba *abandon*, která přeruší právě vykonávanou operaci. To se hodí zvláště při náročných dotazech, kdy vyhledávání trvá příliš dlouho.

11.1.4 Model zabezpečení

Model zabezpečení zajišťuje mechanismus autentizace klienta (prokázání identity) a dává serverům konkrétní práva klienta (autorizace). Jakmile dojde k identifikaci klienta, určí se pomocí informací pro řízení přístupu, zda klient může pracovat s daty, která požaduje. Samotný model LDAP nemá standardizován formát ani možnosti řízení přístupu, proto záleží na jednotlivém využití LDAP a možnostech vývojářů implementovat nejvhodnější model řízení přístupu. Pro účel této práce se budeme zabývat implementací LDAP v databázovém serveru Oracle.

11.2 Oracle Internet Directory

Firma Oracle vytvořila obecně použitelnou adresářovou službu, která kombinuje strukturu LDAPv3 s databázovým serverem Oracle 10g. Nazvala ji Oracle Internet Directory (OID) a mimo jiné umožňuje vývojářům hladkou integraci s jinými adresáři a úložišti identit uživatelů, což společnosti ve výsledku umožní sdílet informace o uživateli bez ohledu na umístění uživatele ([9], volně přeloženo).

Implementace OID sestává ze čtyřech následujících částí ([14], s.326):

- Oracle Directory server – Nástroj zpracovávající přicházející požadavky uživatelů na získání informací o uživatelích či změnu těchto záznamů. Pracuje s protokolem TCP/IP.

- Oracle Directory Replication server – Nástroj, který slouží k replikaci dat uložených v LDAP mezi adresářovými servery.

- Oracle Directory Manager – Klientská aplikace napsaná v Javě, která umožňuje pomocí grafického rozhraní správu OID.

- Ostatní nástroje – V tomto případě se jedná hlavně o správu dat z příkazového řádku či různé jiné nástroje pro správu dat

11.2.1 Ověření uživatelů v OID

Při implementaci OID si můžeme vybrat, jaké možnosti řízení přístupu chceme používat. Na výběr máme tři úrovně ověřování uživatelů, jak lze vidět z tabulky níže. Pro naše účely zvolíme ověření uživatelů na základě certifikátů, protože tato volba poskytuje nejvyšší možnou míru zabezpečení.

Tab. 2 Úrovně ověřování uživatelů

Ověření	**Popis**
Anonymní	Pro přístup do adresáře není vyžadováno jméno ani heslo. Uživatel může pracovat pouze s informacemi určenými pro anonymní uživatele.
Založené na hesle	Klient se identifikuje pomocí DN (distinguished name) a nešifrovaného hesla. Server poté ověří, zda identita (DN) a heslo odpovídají hodnotám v databázi.
Na základě certifikátu (SSLv3)	Klient i server používají certifikáty pro ověření, že identita druhé strany odpovídá skutečnosti.

Zdroj: [14], s. 326

Pro ověřování pomocí certifikátů a protokolu SSL je potřeba tři základní komponenty.

- Certifikát – Komponenta, která je unikátní pro každého uživatele a pomocí které zjistíme informace o uživateli (vlastníkovi certifikátu). Zároveň máme záruku, že tyto informace jsou pravdivé a důvěryhodné. Součástí certifikátu je jméno uživatele (vlastníka), veřejný klíč certifikátu, sériové číslo certifikátu, doba platnosti certifikátu a podpis certifikační autority (CA).

- Certifikační autorita (CA) – Důvěryhodná třetí strana, která vydává certifikáty a odpovídá za ověření informací o vlastníkovi certifikátu. Pokud vše odpovídá skutečnosti, CA podepíše certifikát svým privátním klíčem. V každé síti je nastaven seznam důvěryhodných CA, které bychom měli využít pro ověření našich certifikátů. CA si můžeme vytvořit sami v rámci organizace, ale musíme mít na paměti, že certifikáty ověřené naší CA budou mimo společnost považovány za nedůvěryhodné.

- Peněženka (Wallet) – Tato komponenta slouží jako správce ověřovacích dat vyžadovaných SSL. U serveru Oracle 10g se jedná o privátní klíč, certifikát X.509v3 a seznam důvěryhodných CA.

Níže je popsáno, jak probíhá komunikace uživatele s LDAP adresářovým serverem při použití certifikátů včetně možnosti šifrování komunikace SSL protokolem.

Po navázání spojení mezi uživatelem a serverem na SSL portu (implicitně 636) se obě strany nejprve dohodnou na parametrech spojení (handshake). Server má v sobě uloženo několik sad protokolů pro komunikaci a záleží na uživateli a nastavení serveru, kterou sadu bude využívat.

Jakmile je určena sada protokolů pro komunikaci, server odešle klientovi certifikát. Ten ověří, zda je certifikát podepsaný důvěryhodnou certifikační autoritou. Poté klient odešle svůj certifikát serveru a ten ho ověří stejným způsobem. Veřejné klíče z certifikátů potom klient a server využijí k zahájení komunikace a vytvoření relačního klíče. Tento relační klíč je poté využíván k šifrování a rozšifrování všech zpráv v relaci. Vytvoření relačního klíče znamená, že je ověřena identita uživatele a tento uživatel má přístup k adresáři.

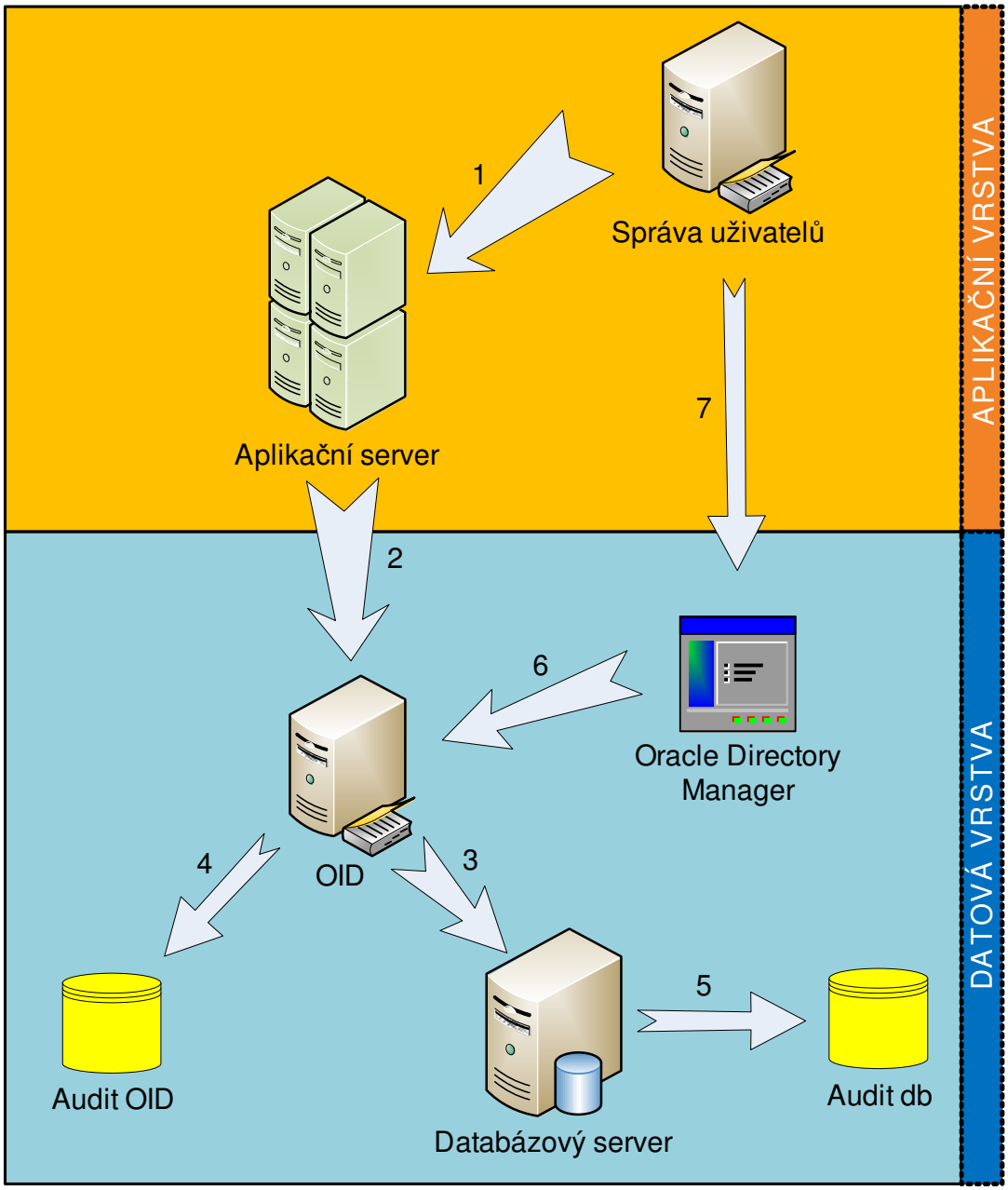
Pokud používáme šifrování SSL protokolem, je každý paket posílaný z klienta na server a zpět kryptograficky zabezpečen pomocí kontrolního součtu. Ten vytvoříme algoritmem MD5 nebo SHA. Úroveň šifrování můžeme volit mezi DES40 a RC4_40, přičemž oba algoritmy využívají klíče o délce 40 bitů. Podle těchto parametrů dochází při vytvoření relace k výběru požadované sady protokolů.

11.2.2 Šifrování hesel a audit v OID

Výhodou OID je možnost uložit hesla k uživatelům v adresáři v šifrované podobě. Implicitní nastavení využívá jednosměrné šifrovací funkce MD4, která pracuje s 128bitovým kontrolním součtem. Uživatel sice obvykle zadává heslo nezašifrované, ale OID ho před ověřením identity zašifruje a porovnává ho se zašifrovaným obrazem hesla uloženým v databázi. Možnosti šifrování hesel však můžeme měnit, na výběr máme několik možností jako algoritmus MD5 (vylepšený MD4) nebo SHA (Secure Hash Algoritmus – 160bitové šifry). Šifrování lze pochopitelně i úplně vypnout.

Pro nastolení a udržení požadované míry zabezpečení je nutné, abychom prováděli audit událostí na databázovém serveru. Pokud budeme používat k ověření uživatelů OID, máme možnost protokolovat i tento adresářový server. Implicitně je sice audit vypnut, ale můžeme celkem jednoduše povolit audit pomocí komponenty Oracle Directory Manager a to na úrovni, kterou budeme požadovat. Události, které můžeme v OID auditovat, jsou uvedeny v tabulce v příloze C včetně masky, která se používá pro jeho povolení v OID.

11.2.3 Schéma řešení



Obr. 11 Využití Oracle Internet Directory

Zdroj: Vlastní tvorba

1. Centrální správa uživatelů autorizovala uživatele pro práci s aplikací a předala požadavek na data aplikačnímu serveru.

2. Aplikace (Business logika) vygeneruje dotaz na data pro databázový server. K tomuto dotazu přidá CustomerID uživatele, který data od aplikace požaduje a celý dotaz odešle na datovou vrstvu, kde jej zpracuje OID.

3. Podle identifikátoru CustomerID OID přiřadí databázové role uživateli, případně požadavek odmítne. Celý dotaz je pak předán samotnému databázovému serveru, který jej zpracuje.

4. O provedených operacích v OID je uchován záznam dle požadované úrovně auditu OID.

5. O prováděných operacích nad daty je uchován záznam dle nastavené úrovně auditu.

6. Oracle Directory Manager slouží jako primární aplikace k plnění OID daty, tedy informacemi o uživatelích a jejich databázových rolích

7. Při využití centrální správy uživatelů ve společnosti je možné synchronizovat informace mezi OID a námi využívaným systémem, což sníží náklady na udržování aktuálního stavu OID.

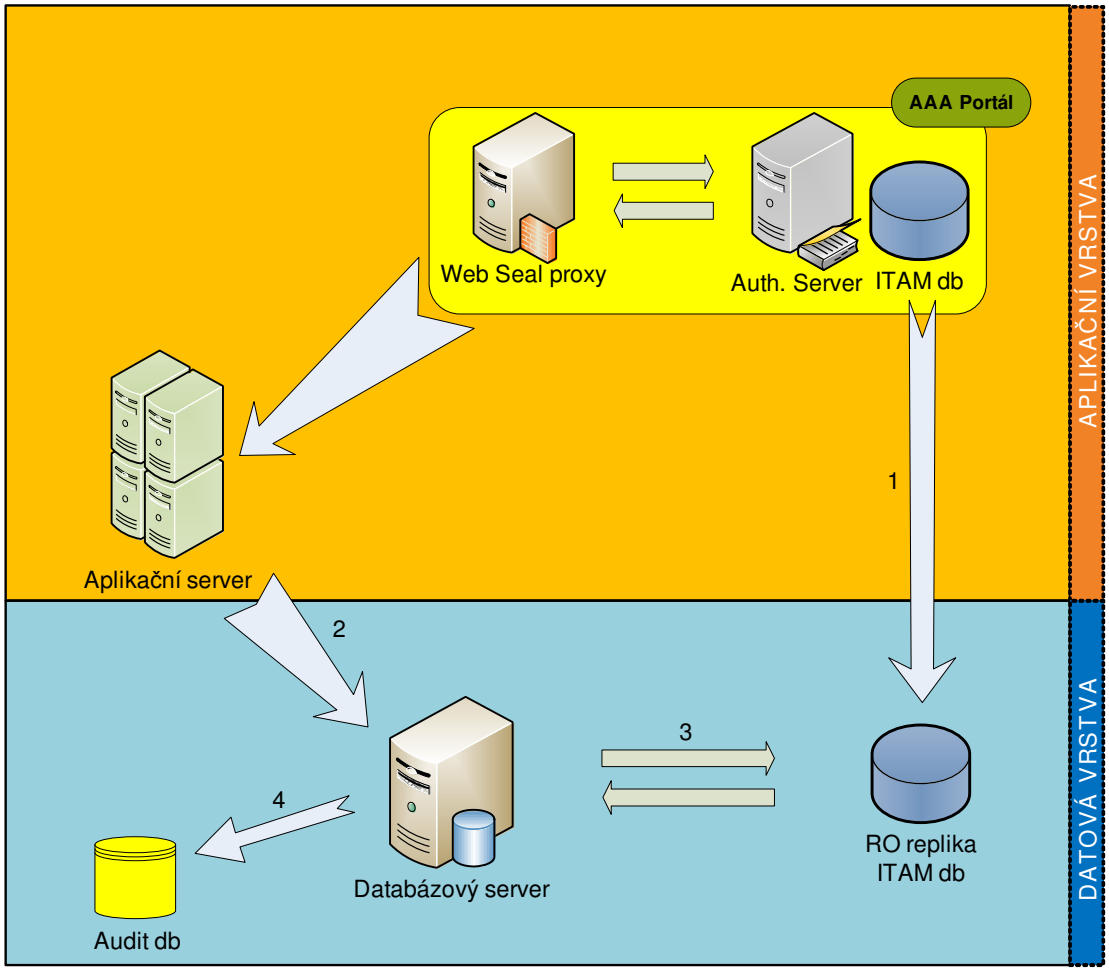
12 VYUŽITÍ ŘEŠENÍ ORGANIZACE STÁTNÍ SPRÁVY

Pokud již ve společnosti existuje fungující architektura zajišťující identifikaci uživatele a přiřazení aplikačních rolí, můžeme ji využít k ověření uživatele na přístupu do databáze. Zde se nabízí otázka, jakým způsobem přenést informace o uživateli a jeho role pro práci s databází na databázový server, když původní nastavení systému s tímto nepočítá. Nejjednodušší možné řešení je přenesení repliky databáze uživatelů na datovou vrstvu. V této databázi budou uloženy informace o uživateli a jeho uživatelské role. Databáze by pak na základě získané identity uživatele od aplikace ověřila platnost této identity a posléze by získala z databáze uživatelů konkrétní přístupová práva k jednotlivým objektům (např. pomocí API rozhraní).

12.1 Replika ITAM db

V našem typickém příkladě využíváme speciálního systému centrální správy uživatelů na prezentační a aplikační vrstvě, AAA Portál. Ten ukládá identitu uživatelů a jeho role do speciální databáze ITAM, postavené na technologii db serveru IBM DB2. K řešení identifikace uživatelů na datové vrstvě potřebujeme vytvořit repliku této databáze uživatelů na této vrstvě tak, aby náš databázový server mohl ověřit identitu klienta z důvěryhodného zdroje.

Tímto způsobem se navíc můžeme zbavit potřeby vytvářet speciální databázi uživatelů a jejich rolí na databázovém serveru. Ať už výše zmíněné využití Virtual Private Database, nebo aplikace LDAP protokolu pomocí Oracle Internet Directory. Výsledným efektem by měla být úspora nákladů spojených s implementací identifikace uživatele na datové vrstvě. Největší překážkou bude kompatibilita databázového serveru s replikou databáze uživatelů, bude tak záležet na konkrétním prostředí, zda se vyplatí přizpůsobení stávajícího systému potřebám databázových serverů, i s ohledem na fyzickou průchodnost jednotlivých vrstev (pokud síťová architektura znemožňuje oboustrannou komunikaci databázových serverů se servery centrální správy uživatelů).



Obr. 12 Využití RO repliky ITAM databáze

Zdroj: Vlastní tvorba

1. Při každé aktualizaci ITAM databáze v AAA Portálu dojde k vytvoření repliky této databáze, která bude určena pouze pro čtení a umístěna na datové vrstvě.

2. Aplikace obslouží požadavek klienta a vytvoří dotaz pro databázi. K dotazu přidá i identifikaci klienta CustomerID, případně i aplikační role pro daného uživatele.

3. Před spuštěním dotazu musí databázový server získat databázové role pro daného uživatele z RO repliky ITAM databáze nebo ověřit role předané aplikací. Každopádně pokud není identita uživatele ověřena z této databáze, db server nemůže požadavek obsloužit a zamítne ho.

4. O prováděných operacích je uchován záznam dle nastavené úrovně auditu.

12.2 Atributové certifikáty

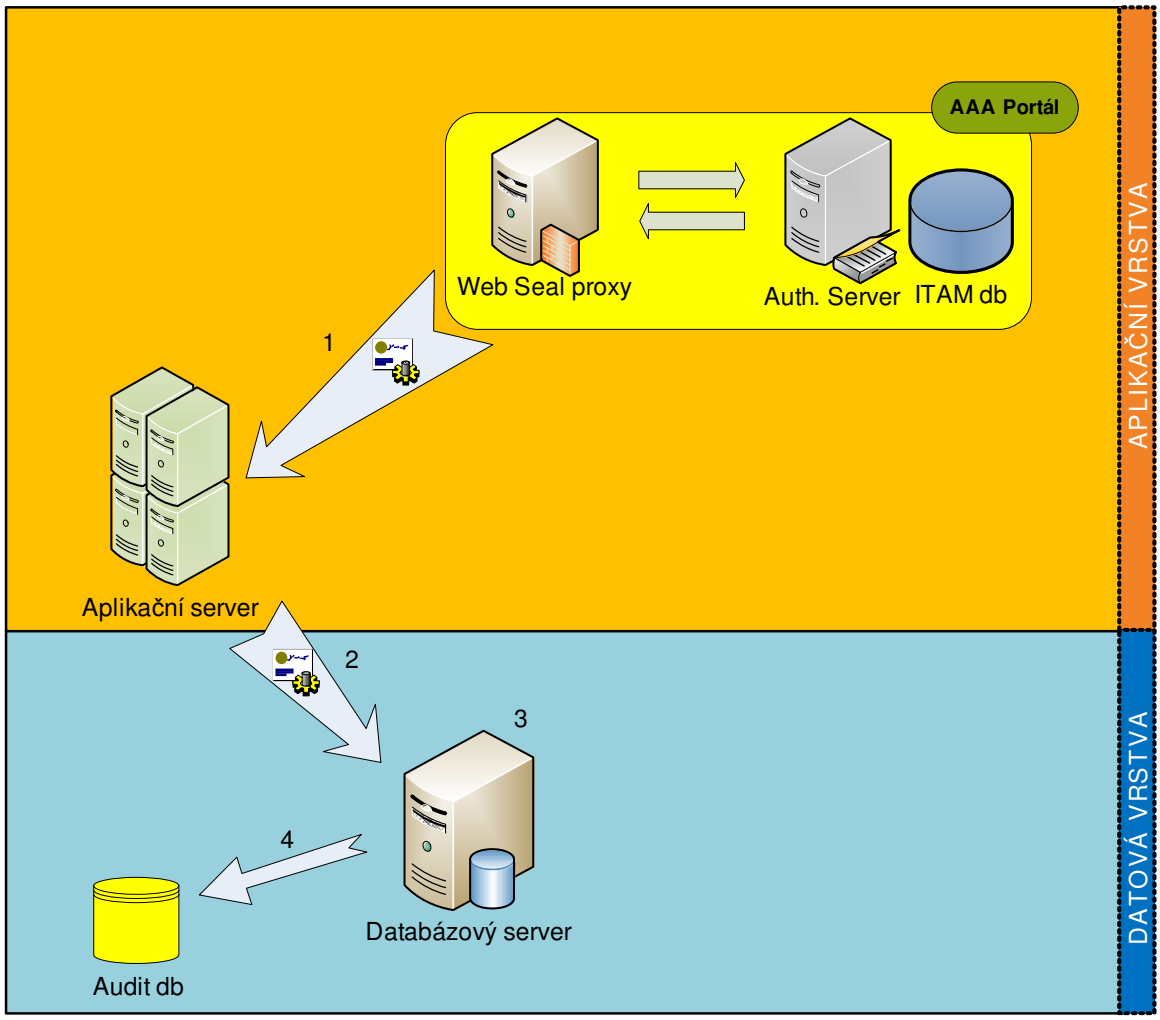
Výše zmíněný způsob přenesení identity klienta a jeho uživatelských rolí na db server má však mnoho úskalí. Velkým problémem je zajištění aktuálního stavu repliky databáze uživatelů na datové vrstvě. Každá změna v přístupových právech by musela být okamžitě přenesena na repliku na datové vrstvě, což vzhledem k vlastnosti Read Only znamená vytvoření nové repliky celé databáze ITAM.

Dalším problémem získávání db rolí z repliky ITAMu je předpokládaný nárůst požadavků na výkon celého systému, což může rychle vést k problémům s dostupností služeb kvůli nedostatku výpočetního výkonu. Pokud snížíme výkonnost systému, stojí za vážnou úvahu, zda má cenu problém zabezpečení datové vrstvy řešit na úkor funkce samotných aplikací (potažmo společnosti).

Princip funkce AAA Portálu zahrnuje i to, že při každém požadavku na aplikačním serveru si volaná aplikace vyžádá od AAA Portálu aplikační role pro daného uživatele. Princip je podobný výše navrženému řešení, proto se zde nabízí možnost získání databázových rolí společně s aplikačními rolemi již při autentizaci a autorizaci klienta před přístupem k aplikaci. Stěžejním problémem je to, abychom zajistili integritu těchto databázových rolí při průchodu aplikací. Jinými slovy, potřebujeme zachovat důvěryhodnost těchto informací tak, aby jim mohl důvěřovat databázový server. Pokud bychom databázové role pouze přidali k dotazu aplikace, dostali bychom se opět na začátek, protože db server by tyto role musel ověřit z vlastní repliky ITAMu na datové vrstvě.

Při komunikaci aplikace s databází ITAM si vyžádá aplikace vlastní aplikační role a navíc ještě vyšle požadavek na databázové role (případně CustomerID, pokud využíváme vlastní uživatele v databázovém serveru), tyto role však dostane aplikace podepsané pomocí atributového certifikátu. Tím se zajistí, že tyto údaje zůstanou na úrovni aplikačních serverů až po databázové servery v nezměněné podobě. Aplikační role však zůstávají nepodepsané. Je to z toho důvodu, že s nimi pracují aplikační servery a ty by zde museli data zase hned rozšifrovat, což by bylo kontraproduktivní.

Je dobré mít na paměti, že atributový certifikát na rozdíl od klasického certifikátu veřejného klíče neobsahuje žádný klíč. Můžeme si také napomoci následující analogií. Certifikát veřejného klíče lze považovat za občanský průkaz (pas), s jehož pomocí je ověřována totožnost uživatele, platí po dlouhé období, je obtížné ho padělat a existuje přísný postup pro autentizaci žadatele o tento certifikát. Atributový certifikát je potom něco jako vstupní vízum, je vydáván odlišnou autoritou a nemá tak dlouhou dobu platnosti jako certifikát veřejného klíče (např. to může být i velice krátké časové období definované v hodinách atd.).Při žádosti o vízum je obvykl předkládán pas pro ověření totožnosti uživatele, vstupní vízum je pak přímo vázáno k tomuto pasu, v pasu je definováno přímo období platnosti víza popř. další podmínky. Atributové certifikáty jsou prostředkem, s jehož pomocí je přenášena informace o oprávněních a to bezpečnou cestou ([12], s. 9).



Obr. 13 Použití atributového certifikátu pro přenos identity

Zdroj: Vlastní tvorba

1. Aplikace si vyžádá aplikační role pro daného uživatele dotazem na databázi ITAM. AAA Portál k těmto aplikačním rolí přidá i databázové role, které zašifruje a podepíše atributovým certifikátem

2. Aplikace vytvoří dotaz na databázi a připojí k němu i atributový certifikát, který zajistí, že s informacemi nebylo manipulováno a jsou důvěryhodné

3. Databázový server ověří podpis na atributovém certifikátu a pokud je správný, obslouží požadavek.

4. O prováděných operacích je uchován záznam dle nastavené úrovně auditu.

13 CELKOVÉ ZHODNOCENÍ MOŽNOSTÍ

V předchozí části této práce bylo prezentováno několik možnosti řešení zabezpečení datové vrstvy v třívrstvé architektuře. Některé technologie lze vzájemně kombinovat, jiné lze efektivně spojit pouze obtížně. Nyní je třeba rozhodnout, která volba představuje nejefektivnější řešení daného problémy s ohledem na finanční náročnost každého řešení.

Čistě z hlediska použité technologie je třeba říci, že každé z prezentovaných řešení lze použít jako efektivní variantu zabezpečení datového zdroje (databáze). Každá varianta splňuje základní předpoklady dostatečného zabezpečení. Podívejme se na konkrétní hrozby, které jsem zmínil při zhodnocení současného stavu a porovnejme je s variantami řešení výše.

13.1 Hrozby:

- *Ve zdrojovém kódu aplikací jsou uloženy přímé SQL dotazy na databázi či jejich šablony a to v nezašifrované podobě.*

  Obecné řešení tohoto problému spočívá v odstranění přímých SQL dotazů ze zdrojového kódu aplikace a přesunutí těchto SQL příkazů do podoby Storage procedures na databázovém serveru tak, aby aplikace předávala pouze parametry pro tyto procedury. Pochopitelně toto řešení vyžaduje dodatečné náklady na tvorbu těchto procedur. Jejich vývoj je nepochybně ekonomicky náročnější než jednoduché vložení SQL dotazu do zdrojového kódu. Toto řešení lze aplikovat u všech variant a lze ho brát jako doplňující prvek k celkovému konceptu zabezpečení dat, protože pokud bychom tuto hrozbu neeliminovali, ohrozili bychom celý zbytek řešení.

- *Možnost jednoduchého zjištění přihlašovacích údajů k databázi z konfiguračního souboru aplikace, ty jsou také uloženy v nezašifrované podobě.*

  Řešení takové situace vyžaduje využití mezičlánku mezi aplikací a databází, který bude zajišťovat přihlášení aplikace k databázi aniž by údaje v konfiguračním souboru mohly prospět útočníkovi. Ideálním příkladem je proxy autentizace.

- *Přímý SQL dotaz na databázový server bez autentizace a autorizace AAA Portálem, tedy bez užití aplikací určených ke komunikaci s databází.*

  Při použití proxy autentizace tuto hrozbu eliminujeme, protože proxy povolí spojení s databází pouze předem nadefinovaným klientům (v tomto případě aplikační vrstva) a bez použití proxy autentizace bude klientovi přístup k databázi odepřen a požadavek bude zamítnut. Výsledkem bude situace, kdy se v případě útoku zúží možné zdroje ohrožení a obrana bude snadnější. Z toho vyplývá, že proxy autentizace je výborný prostředek obrany před útoky, ale pouze v kombinaci s dalšími technologiemi zaměřené na bezpečný přenos identity klienta na databázi.

- *Aplikace s daty může manipulovat dle libosti, proto je možné, že díky chybě v aplikaci dojde ke spuštění procesu, který nepochází od uživatele.*

  Úplná eliminace tohoto rizika je prakticky nemožná a záleží hlavně na schopnostech vývojářů, aby aplikace neprováděla žádné nevyžádané operace. Nicméně nejúčinnější obrana je využití VPD, která omezí práva aplikace na takovou úroveň, která je povolena pro konkrétního uživatele. Tím je splněna potřeba autentizovaného uživatele, protože bez CustomerID VPD dotaz nespustí a odmítne požadavek.

- *Aplikace přistupuje do datové vrstvy přes technologického klienta jako plně privilegovaný uživatel.*

  Tato hrozba je jedna z nejzávažnějších a lze říci, že hlavním úkolem této práce je odstranit právě tuto hrozbu. Všechna výše prezentovaná řešení jsou designována tak, aby aplikace byla nucena přenést identitu klienta na databázi a na základě této identity získala práva k zápisu do databáze. Tak zajistíme, aby aplikace mohla pracovat pouze s daty určenými pro přihlášeného uživatele.

- *Autentizovaný uživatel ověřený AAA portálem se při přístupu k databázi mění v anonymního. Nemožnost auditu samotných uživatelů.*

  Téměř v každé společnosti je třeba hledět na riziko útoku na data společnosti ze dvou stran. Jednak to jsou útoky zvnějšku (přes internet), proti kterým je většina firem bráněna velmi důkladně pomocí firewallů a dalších technologií. Ovšem existuje i další hrozba a to ze strany zaměstnanců společnosti, kteří většinou pracují ve vnitřní síti a mají přístup k aplikacím společnosti. Abychom eliminovali hrozbu ze strany zaměstnanců, je dobré mít k dispozici nástroj, který nám umožní dohledat případného viníka útoku a tím funguje jako dokonalá prevence před útoky zaměstnanců, kteří vědí, že jejich akce jsou sledovány. Takovým nástrojem je audit databáze, který ovšem ke správné funkci potřebuje znát identitu klienta provádějícího změny v databázi. Opět lze tedy říci, že všechna výše prezentovaná řešení umožňují implementaci auditu databáze z rozlišením skutečné identity uživatele.

Je vidět, že ty nejzávažnější hrozby z hlediska bezpečnosti dat dokáží vyřešit všechny varianty, pokud je budeme kombinovat s využitím proxy autentizace. Ta nám zajistí logické oddělení aplikační vrstvy od datové a stanoví prostor k využití technologie určené k přenosu skutečné identity uživatele, abychom mohli omezit práva aplikace na základě databázových rolí či VPD a zároveň provedli záznam o prováděných změnách (audit).

Rozhodnutí v tuto chvíli závisí hlavně na finančních možnostech společnosti a na vůli investovat do této oblasti. Ekonomická náročnost je tak hlavním kritériem výběru nejlepší varianty a proto se na ni podíváme podrobněji.

13.2 Ekonomická kalkulace

V této části textu se podíváme na jednotlivé možnosti řešení, které jsou popsány výše, z hlediska jejich ekonomické náročnosti. Toto hledisko se v našem případě dá rozdělit na dvě odlišné části:

- Náklady na pořízení potřebných technologií (SW licence, výkonnější HW)
- Náklady na pracovní sílu, které jsou potřebné pro úspěšnou implementaci řešení

Přestože má tato práce především poskytnout inspiraci při výběru vhodných technologií pro různá prostředí, při jejich výběru je velmi důležitá ekonomická náročnost těchto řešení. My jsme v textu použili standardní metody aplikované na produkty společnosti Oracle, proto budeme při výpočtu nákladu předpokládat využití těchto produktů v praxi.

13.2.1 Proxy Autentizace

Možnost implementovat mezi databázový server a aplikační vrstvu metodu proxy autentizace je podporována téměř ve všech verzích databázového serveru Oracle Database. Nejlevnější varianta je Oracle Database Standart Edition One za 5000$[1] za licenci (vhodné pro cca 150 uživatelů), dále varianta Standart Edition za 15000$[1] (pro cca 300 uživatelů) a vrchol nabídky představuje Oracle database server Enterprise Edition v ceně 40000$[1] za licenci (určené pro 800 a více uživatelů). Většina obchodních organizací provozující informační systém na databázových serverech Oracle volí verzi Enterprise Edition, která poskytuje komplexní soubor služeb, které společnost potřebuje pro bezchybný provoz informačního systému. Přestože proxy autentizace můžeme použít i u levnějších verzí db

[1] [11]

serveru, v praxi by očesané verze databázového serveru přinesly mnoho problémů, proto bych jejich koupi nedoporučoval nikde, kde se vážně zabývají otázkou bezpečnosti dat.

Nastavení proxy autentizace by měl zvládnout průměrně zkušený administrátor ve společnosti, který umí pracovat s databázovými a aplikačními servery. Náročnost implementace proxy není pro odborníka příliš vysoká, proto lze předpokládat, že by neměla vyžadovat více než 100 hodin práce. Je to i z toho důvodu, že většina administrátorů zná velice dobře principy proxy z firemních sítí a přístupu na internet z nich.

13.2.2 Virtual Private Database

Samotná implementace podpory pro využití VPD je standardní součástí produktu Oracle Database Enterprise Edition v ceně 40000$[2] za licenci. To jasně ukazuje, že levnější verze databázového serveru jsou ochuzeny často právě o komponenty zajišťující ochranu uložených dat, jako např. VPD. Nicméně jak jsme již zmínil v popisu VPD, její konfigurace není právě práce pro laiky a vyžaduje celkem rozsáhlou znalost jazyka PL/SQL a také dobrou představivost, protože pro správu bezpečnostních politik máme k dispozici prakticky pouze příkazový řádek. Oracle samozřejmě nabízí řešení.

Oracle Label Security (OLS) – nástroj umožňující jednoduché nastavení a správu bezpečnostních politik pro VPD. K tomu přidává i několik dalších funkcí spojených se zajištěním více-stupňové ochrany dat v databázi, splňující standardní požadavky obchodních společností i úřadů státní správy ([10], volně přeloženo).

Díky tomuto produktu dosáhneme úspory nákladů na pracovní sílu v podobě administrátorů a vývojářů. I při použití OLS musíme správně navrhnout bezpečnostní politiky a zvolit správný aplikační kontext, ovšem toho dosáhneme především na základě struktury dat, tedy analytické práce a nemusíme být experty na PL/SQL jazyk. Licence OLS znamená investici dalších 10000$[1] k Oracle Database Enterprise Edition.

---

[2] [11]

Otázka nákladů na práci s konfigurací a správou VPD závisí na velikosti spravované databáze a počtu uživatelů, kteří se budou přihlašovat. Pokud budeme uvažovat typické prostředí s max. 1000 uživateli, kteří jsou rozděleny do 10 databázových rolí, implementace VPD pomocí OLS zabere odhadem okolo 300 hodin práce. Přesto jde pouze o odhad, protože pracnost bude značně záležet na kombinaci struktury dat s požadavky na jejich bezpečnost.

13.2.3 Oracle Advanced Security

Pokud budeme chtít využívat pro komunikaci mezi databází a střední vrstvou certifikáty, nejlepším řešením je dokoupení licence na další samostatnou komponentu db serveru Oracle Database Enterprise Edition, a to Oracle Advanced Security (OAS). Tato komponenta zajišťuje pohodlné nastavení a správu šifrování jakýchkoliv dat v databázi, stejně jako šifrování dat přicházejících nebo odcházejících ze serveru. Podporuje většinu protokolů používaných k šifrování dat (PKI, Kerberos, RADIUS) ([7], volně přeloženo).

Cena dle ceníku Oracle je stanovena na 10000$[1] za licenci, ale pouze pokud máte již zakoupenu licenci Oracle Database Enterprise Edition. Pokud budeme uvažovat náklady na pořízení licence samotného db serveru (40000$[3]), nejeví se tato částka jako nijak závratná. Naopak s přihlédnutím k možnostem, které nám zabezpečení dat pomocí šifrování dat a využívání certifikátů nabízí, zdá se tato komponenta jako velice vhodný doplněk k samotnému databázovému serveru, zvláště u společností kladoucí velký důraz na zabezpečení firemních dat, protože právě šifrování dat a použití certifikátů je velice výkonný a efektivní způsob zabezpečení dat v databázi.

Konfigurace této komponenty a samotná implementace certifikátů do procesu autentizace se na první pohled nezdá příliš složitá. Bohužel osobní zkušenost ukazuje, že práce s certifikáty vyžaduje velikou preciznost v nastavení všech součástí systému a znalost jejich nastavení není u administrátorů příliš běžná. Proto je při kalkulaci nákladů třeba brát v úvahu možnou potřebu školení pověřených zaměstnanců (to platí i v předešlých případech, ovšem u této varianty je to nejpravděpodobnější). Naopak je zde ještě jeden

[3] [11]

pohled, který hovoří pro tuto investici. Použití certifikátů pro ověření identity je bezpochyby moderní technologií v této oblasti a jejich využití se bude rozhodně zvyšovat. Proto si dovolím tvrdit, že tato technologie se v současné době jeví jako nejpokrokovější a její úspěšná implementace na databázi může poskytnout užitečné zkušenosti při implementaci certifikátů v jiných částech podnikového informačního systému či naopak můžeme využít již předešlých zkušeností s implementací certifikátů, např. elektronický podpis objednávek.

13.2.4 Oracle Internet Directory

Implementace LDAP adresářové služby pro přístup k databázi vypadá jako velmi elegantní a funkční řešení. Nicméně pokud využíváme jiného systému pro správu uživatelů pro ostatní komponenty informačního systému, je samotné použití LDAP nad databázovým serverem dost drahé řešení. Je to z toho důvodu, že většina společností dodává tuto technologii jako součást komplexního balíku určeného ke správě uživatelů společnosti, proto podporuje mnoho dalších funkcí, které nejsou nezbytné pro přístup na databázový server. Konkrétně implementace Oracle internet directory (OID) je možné pouze jako součást komplexního produktu Oracle Identity Manager ([8], volně přeloženo).

Integraci produktu do uceleného balíku odpovídá i cena 40000$ za připojení jedné komponenty (např. db serveru).Dále je třeba zaplatit za jednotlivé uživatele systému. 60$ za uživatele – zaměstnance a 5$ za externího uživatele[4]. Pokud tedy předpokládáme 1000 uživatelů, dostáváme se ke 100000$, což již pochopitelně není zanedbatelná částka. Při použití Identity Manageru na ostatní komponenty IS se sice zvýší náklady na připojení dalších komponent, ale ve výsledku ušetříme za administraci jednotného systému autentizace uživatelů. To platí opět i obráceně, vyplatí se přizpůsobit stávající produkty využívající LDAP protokolu a upravit je k autentizaci uživatelů na datové vrstvě.

Administrace OID, nebo LDAP adresářové služby obecně, je poměrně snadná a intuitivní záležitost a může ji vykonávat každý administrátor. Pokud již ve společnosti funguje nějaká komponenta na podobném principu, úkolem administrátora bude hlavně dohlížet na

[4] [11]

synchronizaci mezi jednotlivými částmi, která navíc bude probíhat automaticky. Pracnost zavedení OID by neměla přesáhnout 100 – 200 hodin práce, opět v závislosti na počtu uživatelů a případné existenci podobné adresářové služby uživatelů.

13.2.5 Porovnání nákladnosti

Pro představu o rozdílech v nákladech na implementaci jednotlivých řešení prezentovaných výše je dobré rozdělit náklady do tří skupin. První skupinou jsou náklady na lidské zdroje (HR), druhou náklady na pořízení a aktualizaci licencí k použitému softwaru a nakonec náklady na posílení HW vybavení potřebného k plynulé funkci systému.

Tab. 3 Porovnání nákladnosti prezentovaných řešení

Typ řešení	**Náklady na HR**	**Náklady na SW**	**Náklady na HW**
Proxy autentizace	Nízké - Střední	Nízké	Vysoké
Virtual Private Database	Střední - Vysoké	Nízké - Střední	Střední
Oracle Advanced Security	Vysoké	Nízké - Střední	Střední
Oracle Internet Directory	Nízké - Střední	Vysoké	Střední

Zdroj: Vlastní tvorba

14 ZÁVĚR

Hlavním cílem této práce bylo nalézt různé varianty řešení, které by umožnily vyhovět dnešním nárokům na bezpečnost dat. Při použití třívrstvé architektury vývojáři často opomíjejí zabezpečení samotné databáze (datová vrstva) a tento nedostatek je třeba při vývoji moderních aplikací odstranit a implementovat nové technologie, které zajistí dostatečnou ochranu před útoky (včetně těch neúmyslných).

Pro ilustraci modelového prostředí byla použita společnost státní správy, která již využívá systém pro autentizaci uživatelů na rozhraní klientské (prezentační) a aplikační vrstvy – AAA Portál. V takovém případě je nejlepší variantou nezatěžovat rozpočet společnosti náklady na vývoj speciálního řešení a efektivnější řešení je přenos informací o identitě uživatele z tohoto systému na databázový server. Z tohoto pohledu se tak jeví jako nejefektivnější využití AAA Portálu i ke správě databázových rolí, které portál předá aplikaci současně s aplikačními rolemi. Databázové role však budou pro aplikaci nečitelné, protože budou podepsány atributovým certifikátem tak, aby tyto informace mohl rozšifrovat pouze databázový server a zároveň těmto informacím důvěřoval.

Výsledky této práce lze také aplikovat do obecného prostředí, kde neexistuje takový komplexní systém zabezpečení jako AAA Portál. Obecně lze totiž říci, že drtivá většina společností provozujících podnikový informační systém využívá centrální autority spravující uživatele a jejich role v podnikových aplikacích (např. Microsoft Active Directory). Nejlevnějším řešením proto bude přiřazení speciálních databázových rolí těmto uživatelům a přenos těchto rolí na datovou vrstvu pomocí atributového certifikátu. Pokud by společnost takový systém neprovozovala, ukázali jsme si řešení využívající vlastní adresář uživatelů a jejich rolí (LDAP v Oracle Internet Directory). Toto řešení je ekonomicky výrazně náročnější, ale odráží v sobě i možnou využitelnost na jiných místech informačního systému společnosti nežli je rozhraní aplikační a datové vrstvy.

Tato práce neobsahuje konkrétní praktické řešení dané problematiky, protože aplikace do praxe je finančně poměrně náročná záležitost a většina společností zatím není ochotna

investovat do této oblasti (projekt v modelovém prostředí společnosti státní správy na toto téma byl zamítnut…). Bohužel dokud nebude zabezpečení datové vrstvy bráno jako konkurenční výhoda či dokonce nutný standart, společnosti budou velmi těžko hledat prostředky na řešení této situace. Navíc vidina celosvětové ekonomické krize oblast bezpečnosti dat obecně poškodí poměrně výrazně, protože pro ekonomy je tato oblast mezi prvními v řadě úspor nákladů. Nicméně v blízké budoucnosti je jisté, že bezpečnost dat bude stále aktuálnější problém a je pouze otázkou času, než budou muset vývojáři aplikací počítat s komplexním systémem zabezpečení dat na všech úrovních, včetně té poslední, datové.

Každá společnost, stejně jako vývojář aplikací, může v této práci nalézt vodítka, která ho provedou problematikou zabezpečení databáze a přenos identity klienta na databázový server a pomůžou v rozhodování, kterým směrem se ubírat. Většina praktických problémů je řešena pomocí produktů společnosti Oracle, protože databázový server Oracle je jeden z nejrozšířenějších v podnikových informačních systémech. Jak jsem již ale zmínil, na trhu existuje několik velkých korporací (Microsoft, IBM, SAP), v jejichž produktovém portfoliu jsou komponenty využívajících stejných principů a metod jako produkty Oracle zmíněné v této práci. Prezentované závěry tak lze aplikovat i na mnoho jiných produktů a využít konkrétních možností, které nám pomohou vyřešit problémy praktické implementace.

Jak jsme již zmínil, hloubka zabezpečení dat společnosti nejvíce závisí na ochotě managementu investovat do této oblasti, protože neexistují jednotné směrnice ani instituce, které by vymahatelně stanovily jasná pravidla a požadavky. Využitelnost této práce tak zůstává závislá na obecném povědomí osob zodpovědných za bezpečnost dat v podniku a jejich ochotě se touto problematikou zabývat. Pokud ve společnosti padne rozhodnutí investovat do této oblasti, tato práce by měla sloužit jako vodítko ke správnému řešení na základě konkrétních možností dané společnosti.

15 ZDROJE:

15.1 Citace

1 Interní dokumenty Siemens

2 KNOX, D. *Effective Oracle Database 10g Security by Design.* 1. vyd. McGraw-Hill/Osborne, 2004. 560 s. ISBN 978-0072231304

3 KOMATINENI,S. Qualities of a Good Middle-Tier Architecture. *OnJava.com* [online]. 2003 [cit. 2008-11-15]. Dostupný z WWW: <http://www.onjava.com/pub/a/onjava/2003/10/01/middletier.html>

4 MAASSEN A., SCHOENEN M. aj. SAP R/3 Kompletní průvodce, 1. vyd. Brno: Computer Press, 2007. 724 s. ISBN 978-80-251-1750-7

5 NORTHCUTT S. a ZELTSER L. Bezpečnost počítačových sítí – Kompletní průvodce návrhem, implementací a údržbou zabezpečené sítě. 1. vyd. Brno: CP Books, 2005. 568 s. ISBN 80-251-0697-7

6 *Open Web Application Security Project* [online], 2008 [cit. 2008-10-13]. Dostupný z WWW: <www.owasp.org>

7 *Oracle Advanced Security* [online]. Oracle Corporation, 2008 [cit. 2008-11-22]. Dostupný z WWW: <http://www.oracle.com/database/advanced-security.html>

8 *Oracle Identity Manager* [online]. Oracle Corporation, 2008 [cit. 2008-11-22]. Dostupný z WWW: <http://www.oracle.com/products/middleware/identity-management/identity-manager.html>

9 *Oracle Internet Directory 10.1.4.2 Data Sheet* [online]. Oracle, 2008[cit. 2008-11-15]. Dostupný z WWW: <http://www.oracle.com/technology/products/oid/pdf/internet_directory_ds_10gr3.pdf>

10 *Oracle Label Security* [online]. Oracle Corporation, 2008 [cit. 2008-11-22]. Dostupný z WWW: <http://www.oracle.com/database/label-security.html>

11 *Oracle Technology Global Price List* [online]. Oracle Corporation, 2008 [cit. 2008-11-22]. Dostupný z WWW: <http://www.oracle.com/corporate/pricing/eplext.pdf>

12 PINKAVA, J. Atributové certifikáty – profil dle rfc.3281. *Crypto-world* [online]. 2003, roč. 3, č. 5 [cit. 2008-12-02].. Dostupný z WWW: <http://crypto-world.info/casop5/crypto05_03.pdf>

13 RUSSELL, J. *Oracle9*i *Application Developer's Guide* [online]. 2nd release, Oracle Corporation, 2002 [cit. 2008-11-12]. A96590-01. Dostupný z WWW: <http://download.oracle.com/docs/cd/B10500_01/appdev.920/a96590/adgsec03.htm>

14 THERIAULT M., NEWMAN A. *Bezpečnost v Oracle.* 1. vyd. Brno: Computer Press, 2004.516 s. ISBN 80-722-6979-8

15 TŮMA, P. *Co nového přinášejí moderní softwarové architektury?* AUTOMA: Odborný časopis pro automatizační techniku. Praha: 2004, roč. 10., č.6. ISSN 1210-9592

15.2 Bibliografie

1 ADÁMEK, P. *Ročníkový projekt.* [Ročníkový projekt.] Brno: Vysoké učení technické v Brně – Fakulta elektrotechniky a informatiky, 1999 [cit. 2008-09-25]. Dostupný z WWW: <http://www.herkules.cz/doc/rp/rp.html>

2 Business Logic. *Wikipedia, The free encyclopedia* [online]. 2008 [cit. 2008-11-30]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Business_logic>

3 DOLEŽAL,Z. Audit DB. [Referát.] Praha: Univerzita Karlova v Praze – Matematicko -fyzikální fakulta, 2006 [cit. 2008-11-12]. Dostupný z WWW: <http://www.ms.mff.cuni.cz/~kopecky/vyuka/oracle2/0607/audit.pdf>

4 HEMPHILL,M. A closer look at the Application Layer. *MIS for the Information Age* [online]. 2007 [cit. 2008-11-15]. Dostupný z WWW: <http://exonous.typepad.com/mis/2007/11/a-closer-look-a.html>

5 NANDA, A. Keeping Information Private with VPD. *Oracle Magazíne* [online]. 2004, č. 3 [cit. 2008-11-12]. Dostupný z WWW: <http://www.oracle.com/technology/oramag/oracle/04-mar/o24tech_security.html>

6 *Oracle9i database online documentation* [online]. Oracle. 2006 [cit. 2008-11-22]. Dostupný z WWW: <http://download-uk.oracle.com/docs/cd/B10501_01/server.920/a96521/audit.htm>

7 *Proxy Authentization in Oracle* [online]. Oracle Technology Network, 2008 [cit. 2008-11-15]. Dostupný z WWW: <http://www.oracle.com/technology/deploy/security/database-security/proxy/index.html>

8 URLWIN, J. Oracle Proxy Authentication, *nntp.perl.org* [online], 2008 [cit. 2008-11-15].. Dostupný z WWW: <http://www.nntp.perl.org/group/perl.dbi.dev/2008/11/msg5517.html>

9 YANG, J. Boosting Your .NET Application Performance, *Developer vision* [online]. 2004, s.2 [cit. 2008-11-22]. Dostupný z WWW: <http://www.developerfusion.com/article/3058/boosting-your-net-application-performance/2/>

16 SEZNAM PŘÍLOH

Označení	Název	Rozsah
Příloha A	Parametry funkce SYS_CONTEXT	2 str.
Příloha B	Parametry procedur v DBMS_RLS	1 str.
Příloha C	Události pro audit v OID	1 str.

Příloha A – Parametry funkce SYS_CONTEXT

Atribut	Vrácená hodnota
AUTHENTIZATION_DATA	Data použitá pro ověření přihlášeného uživatele.
AUTHENTIZATION_TYPE	Metoda použitá při ověření uživatele. Možné hodnoty jsou: DATABASE - ověření na základě uživatelského jména a hesla, OS - ověření externího uživatele operačním systémem, NETWORK - síťový protokol nebo ověření, ANO PROXY - ověření přes zprostředkovatele OCI spojení.
DB_JOB_ID	Pokud pro aktuální relaci běží proces na pozadí, vrací jeho ID.
CLIENT_INFO	Používá se pro uložení informací ve spojení s balíkem DBMS_APPLICATION_INFO. Parametr vrací 64 bytů informací o relaci uživatele.
CURRENT_SCHEMA	Jméno implicitního schématu, které se používá jako aktuální schéma.
CURRENT_USER	Jméno uživatele, s jehož přístupovými právy relace pracuje.
CURRENT_USERID	ID uživatele, s jehož přístupovými právy relace pracuje.
DB_DOMAIN	Jméno domény určené iniciačním parametrem DB_DOMAIN.
DB_NAME	Jméno databáze určené iniciačním parametrem DB_DOMAIN.
ENTRYID	Identifikátor relace pro audit. Je nastaven, pokud je v souboru parametrů nastaven parametr AUDIT_TRAIL na hodnotu TRUE.
EXTERNAL_NAME	Externí jméno databázového uživatele. Toto rozlišovací jméno uložené v uživatelském certifikátu je vráceno při ověření relace přes SSL pomocí certifikátů v.503.
FG_JOB_ID	Pokud je pro relaci spuštěn proces na popředí, vrací jeho ID.

HOST	Jméno počítače, ze kterého je uživatel připojen.
INSTANCE	Identifikační číslo aktuální instance.
ISDBA	Pokud je zapnuta role DBA, vrací hodnotu TRUE. Jinak vrací FALSE.
NETWORK_PROTOCOL	Síťový protokol, který se pro danou relaci používá.
OS_USERS	Jméno uživatele u procesu v operačním systému na klientovi, který zahájil relaci s databází.
PROXY_USERS	Jméno databázového uživatele, který navázal relaci v zastoupení uživatele SESSION_USER.
PROXY_USERID	ID databázového uživatele, který navázal relaci v zastoupení uživatele SESSION_USER.
SESSION_USER	Databázové jméno uživatele, jako který byl uživatel relace ověřen. Hodnota se v rámci relace nemění.
SESSION_USERID	Identifikátor uživatele, jako který byl uživatel relace ověřen.
SESSIONID	Identifikátor relace pro audit.
TERMINAL	Identifikátor operačního systému na klientovi pro aktuální relaci.

Příloha B – Parametry procedur v DBMS_RLS

Parametr	Procedura	Popis
object_schema	ADD_POLICY DROP_POLICY REFRESH_POLICY ENABLE_POLICY	Jméno schématu, ve kterém leží tabulka nebo pohled
object_name	ADD_POLICY DROP_POLICY REFRESH_POLICY ENABLE_POLICY	Jméno tabulky nebo pohledu
policy_name	ADD_POLICY DROP_POLICY REFRESH_POLICY ENABLE_POLICY	Jméno politiky, která má být k tabulce přidána nebo odebrána. Musí být unikátní v rámci tabulky
function_schema	ADD_POLICY	Schéma, ve kterém leží funkce implementující politiku.
policy_function	ADD_POLICY	Jméno funkce, která generuje pro politiku prediktát. Pokud je funkce definovaná jako součást balíku, musí být zadáno i jméno balíku.
statement_types	ADD_POLICY	Typy příkazů, na které se politika vztahuje. Může to být libovolná kombinace příkazů select, insert, update a delete. Není-li určeno jinak, vztahuje se na všechny typy příkazů.
update_check	ADD_POLICY	Je-li nastaveno na hodnotu TRUE, bude hodnota kontrolována proti bezpečnostním politikám až po provedení příkazu insert nebo update. Nepovinný parametr
enable	ADD_POLICY ENABLE_POLICY	Indikuje, zda je politika po přiřazení povolena. Implicitní hodnota je TRUE

Příloha C – Události pro audit v OID

Událost	Popis	Maska
Superuser login	Připojení superuživatele na server (úspěšné i neúspěšné)	0x0001
Schema element (přidání nebo nahrazení)	Přidání nového elementu do schématu (úspěšné i neúspěšné)	0x0002
Schema element (smazání)	Smazání schématu (úspěšné i neúspěšné)	0x0004
Bind	Neúspěšné pokusy o přihlášení	0x0008
Access Violation	Odepření přístupu na základě ACP	0x0010
DSE modification	Změna záznamu DSE (úspěšné i neúspěšné)	0x0020
Replication login	Ověření replikačního serveru (úspěšné i neúspěšné)	0x0040
ACL modification	Změna ACP	0x0080
User password modification	Změna hesla pro uživatele (úspěšné i neúspěšné)	0x0100
Add	Operace LdapAdd (úspěšné i neúspěšné)	0x0200
Delete	Operace LdapDelete (úspěšné i neúspěšné)	0x0400
Modify	Operace LdapModify (úspěšné i neúspěšné)	0x0800
Modify DN	Operace LdapModifyDN (úspěšné i neúspěšné)	0x1000