View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DSpace at VSB Technical University of Ostrava

VŠB - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Zabezpečení systému eLogika

Security of eLogika System

2010

Bc. Vít Zátopek

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7.5.2010 …………………………..

Vít Zátopek

Poděkování

Chtěl bych tímto poděkovat vedoucímu bakalářské práce Mgr. Marku Menšíkovi Ph.D. za odbornou pomoc a konzultace při řešení problémů.

Abstrakt a klíčová slova

Abstrakt

Tato diplomová práce popisuje metody, kterými lze dosáhnout vysokého stupně zabezpečení dat, uložených v databázovém systému Microsoft SQL Server 2008. Součástí popisu metod je i ukázka jejich implementace v rámci e-learningového systému eLogika. Práce dále popisuje návrh a implementaci systému pro odhalování neoprávněného přístupu k datům, kterými systém eLogika disponuje.

Klíčová slova

Cross Site Request Forgery, Cross Site Scripting, SQL Injection, SQL Server, databáze, C#, Transact-SQL, e-learning, šifrování, bezpečnost

Abstract and key words

Abstract

This diploma thesis describes methods for reaching high level of data security. Data are saved in database system Microsoft SQL Server 2008. Description of methods even contains example of their implementation in a scope of e-learning system eLogika. Thesis further describes design and implementation of system for detecting unauthorised access to data which system eLogika has.

Key words

Cross Site Request Forgery, Cross Site Scripting, SQL Injection, SQL Server, database, C#, Transact-SQL, e-learning, encryption, security

Seznam použitých symbolů a zkratek

CLR	Common Language Runtime
CSRF	Cross Site Request Forgery
IIS	Internet Information Services
IP	Internet Protocol
LINQ	Language INtegrated Query
LMS	learning management system
OS	Operation System
SQL	Structured Query Language
T-SQL	Transact SQL
URL	Uniform Resource Locator
XSS	Cross Site Scripting

Obsah

1 Úvod ..... - 1 -

2 Systém eLogika ..... - 2 -

2.1 Implementační prostředí ..... - 2 -

2.2 Uživatelé systému eLogika ..... - 2 -

3 Správa přístupu k databázi systému eLogika ..... - 6 -

3.1 Uživatelé aplikace a databázového serveru systému eLogika ..... - 6 -

3.2 Mapování uživatelů systému eLogika na uživatele databáze ..... - 8 -

3.3 Správa rolí ..... - 15 -

3.4 Použití koncových bodů ..... - 21 -

3.5 Výchozí uživatelé SQL Serveru ..... - 23 -

3.6 Zálohování databáze ..... - 24 -

3.7 Transparentní šifrování dat ..... - 28 -

4 SQL Audit ..... - 30 -

4.1 SQL Audit v MSSQL 2008 ..... - 31 -

4.2 Nastavení auditu pro potřeby systému eLogika ..... - 32 -

4.3 Příprava dat pro zpracování výstupu auditu ..... - 36 -

4.4 Analýza auditních záznamů ..... - 45 -

4.5 Ukázka výsledné aplikace ..... - 51 -

5 Bezpečnost webových aplikací ..... - 55 -

5.1 Nejčastější útoky na webové aplikace ..... - 55 -

6 Závěr ..... - 60 -

Použitá literatura ..... - 61 -

Seznam obrázků

Obrázek 2.1 - Diagram případů užití aktéra Administrátor - 3 -
Obrázek 2.2 - Diagram případů užití aktéra Tajemník - 4 -
Obrázek 2.3 - Diagram případů užití aktéra Garant - 4 -
Obrázek 2.4 - Diagram případů užití aktéra Tutor - 5 -
Obrázek 2.5 - Diagram případů užití aktéra Student - 5 -
Obrázek 3.1 - Šifrování dat metodou CBC - 11 -
Obrázek 3.2 - Proces příhlášení - 15 -
Obrázek 3.3 - Rozdíl v typech záloh databáze - 26 -
Obrázek 3.4 - Vytvoření kroku pro úkol SQL Server Agenta - 28 -
Obrázek 3.5 - Vytvoření časového plánu pro úkol SQL Server Agenta - 28 -
Obrázek 4.1 - Struktura auditu SQL Serveru - 31 -
Obrázek 4.2 – Různé varianty posloupnosti událostí - 50 -
Obrázek 4.3 - Seznam uživatelů se záznamem v logu - 51 -
Obrázek 4.4 - Seznam aktivit konkrétního uživatele - 51 -
Obrázek 4.5 - Detaily aktivity - 52 -
Obrázek 4.6 - Výpis s nestandardním chováním uživatele v aplikaci - 52 -
Obrázek 4.7 - Vástup analýzy s nalezeným neoprávněným přístiupem k datům - 53 -

Seznam tabulek

Tabulka 4.1 - Možné posloupnosti při různém stavu systému - 39 -
Tabulka 4.2 - Mapování záznamů z databáze na objekty - 46 -

1 Úvod

S velkým rozmachem internetových aplikací, které jsou stále oblíbenější a používanější, roste rovněž zájem o data, které aplikace obsahují. Důvody pro jejich získání mohou být různé. Od těch nejprostších jako je vlastní obohacení na základě znalosti informací vycházejících z odcizených dat, přes finanční obohacení plynoucí z jejich prodeje, až po jejich zneužití k dalšímu poškození uživatelů, či samotné aplikace. Je tedy v zájmu provozovatelů webových aplikací, aby předcházeli možnému úniku dat.

Náplní této práce je návrh a implementace metod zajišťujících bezpečnost dat v e-learningovém systému eLogika. Vzhledem k tomu, že e-learningové systémy neslouží jen jako podpůrné nástroje při běžné výuce, ale probíhá jejich prostřednictvím i přezkušování studentů, je zajištění bezpečnosti dat jednou z hlavních podmínek, které jsou na ně kladeny.

Po představení systému eLogika, následuje kapitola věnovaná přístupu k datům, uloženým v relační databázi. Obsahem této kapitoly je shrnutí obecných metod využívaných ve webových aplikacích a konkrétní návrh a zpracování přístupu k datům pro systém eLogika. U něj je kladen důraz na minimalizaci následků možného napadení celé aplikace a zároveň poskytnutí prostředků pro další bezpečnostní prvky.

Čtvrtá kapitola se zabývá návrhem a implementací systému pro odhalování neoprávněné manipulace s daty, kterými systém eLogika disponuje. Tento systém je navržen tak, aby zachytil útok přicházející nejen z prostředí samotné webové aplikace, ale i z jakéhokoliv jiného zdroje. Hlavní myšlenka, na které je systém postaven, tkví ve sledování, jaká data jednotliví uživatelé požadují. Na jejich základě je vyhodnoceno, zda jde o běžné chování při práci s eLogikou, nebo jde o možné napadení.

Poslední kapitola této práce, přibližuje nejčastější typy útoků na webové aplikace a nabízí možná opatření, která slouží ke snížení možnosti provedení útoků, či k jejich úplnému znemožnění.

2 Systém eLogika

Jedním z cílů této práce je analyzovat možnosti zabezpečení systému eLogika, což je systém pro řízení výuky (LMS), se zaměřením na potřeby kurzů z oboru logiky. Elogika je provozována jako webová aplikace, z čehož plynou také možné bezpečnostní ohrožení, na které se zaměřím v této práci.

LMS je aplikace řešící administrativu a organizaci výuky v rámci e-learningu (1). Jde tedy o aplikaci umožňující vytváření a správu kurzů, výukových materiálů, studijních plánů a dalších objektů spjatých s výukou. Součástí LMS je také správa podkladů pro ověřování nabytých znalostí studentů, kteří jsou rovněž v systému evidováni.

E-learning je pak forma vzdělávání založená na využívání informačních a komunikačních technologií (2).

Hlavní odlišností eLogiky od jiných LMS je možnost kontrolovat celý postup řešení zadaného příkladu, a ne pouze správnost jeho výsledku. Na základě správnosti jednotlivých kroků, lze pak dle zadaného vzorce automaticky určit odpovídající ohodnocení dané otázky. Kvůli potřebám korektního zobrazení materiálů obsahujících logické formule, byla implementována podpora zadávání logických symbolů pomocí zástupných znaků, případně pomocí maker programu TeX.

2.1 Implementační prostředí

Systém eLogika je webová aplikace postavená na technologii ASP.NET, která je součástí .NET Frameworku, což je prostředí pro běh aplikací, vyvinuté společností Microsoft Corporation. Výhodou ASP.NET je, že je založen na vykonávacím prostředí CLR, které je společné pro celý .NET Framework. Nebudu zabíhat do detailů týkajících se .NET Frameworku, podstatné je však to, že díky CLR lze nejen webové aplikace psát v jakémkoliv programovacím jazyce, který CLR podporuje. Pro eLogiku byl zvolen objektově orientovaný jazyk C#.

Pro uchování dat byl vybrán relační databázový systém Microsoft SQL Server 2008. Ten je k dispozici v sedmi různých verzích, přičemž jen ta nejvyšší, Enterprise, disponuje nadstandardními funkcemi a nástroji, které nacházejí uplatnění i v rámci eLogiky. Jsou to hlavně funkce z oblasti bezpečnosti a data miningu. Z tohoto důvodu byla zvolena tato verze MS SQL Serveru.

Operační systém pro server, na kterém je eLogika provozována, byl vybrán s ohledem na použité technologie, uvedené výše. Konečná volba padla na Microsoft Windows Server 2003. Pro samotný běh webových aplikací je součástí OS webový server IIS 6.0.

Vývojové nástroje jsou voleny opět na základě použitých technologií. Pro vývoj webové aplikace je použito vývojové prostředí Microsoft Visual Studio 2008, pro správu databázového serveru pak SQL Server Management Studio.

2.2 Uživatelé systému eLogika

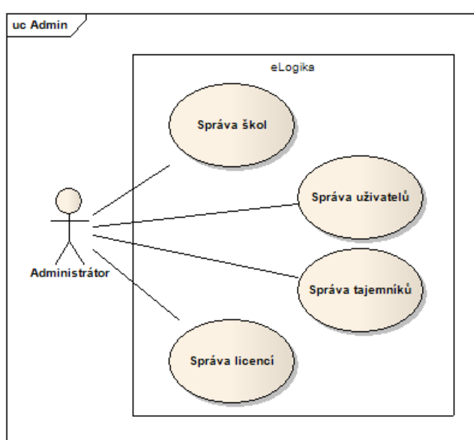
Uživatelem eLogiky je každý člověk, který má v systému vytvořen svůj účet. Samotná existence účtu ovšem neumožňuje práci se systémem, jelikož takový uživatel nemá oprávnění na žádné operace v rámci systému. Aby uživatel tyto oprávnění získal, musí být členem některé z rolí.

V systému eLogika je zavedeno pět rolí, které odrážejí reálnou strukturu odpovědnosti či oprávnění v rámci univerzity. Každý uživatel, je členem jedné nebo více rolí. Současně může uživatel vystupovat pouze v rámci jedné role, kterou si volí při přihlášení k systému.

2.2.1 Role administrátor

Administrátor je role, která má v rámci celého systému eLogika nejvyšší oprávnění. Uživatel v této roli se stará o správu škol, které mají udělenu licenci pro práci se systémem. Současně má jako jediný uživatel možnost spravovat všechny uživatelské účty v systému.

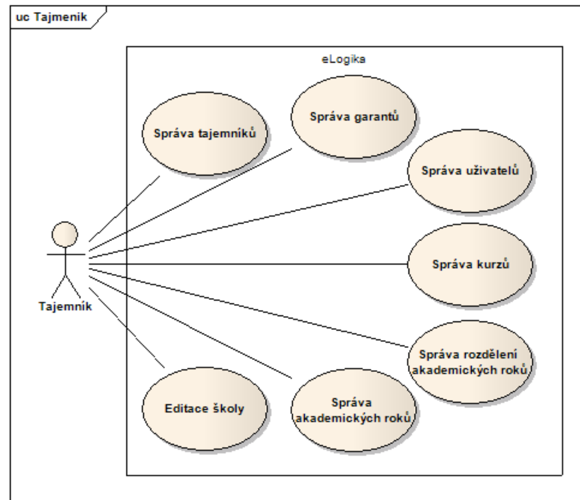
Možnosti, které má uživatel v roli administrátor při práci se systémem eLogika, jsou znázorněny v diagramu případu užití. Jde o část diagramu případu užití nulté úrovně, který je příliš rozsáhlý, a proto jsem jej rozdělil podle jednotlivých aktérů.



Obrázek 2.1 - Diagram případů užití aktéra Administrátor

2.2.2 Role tajemník

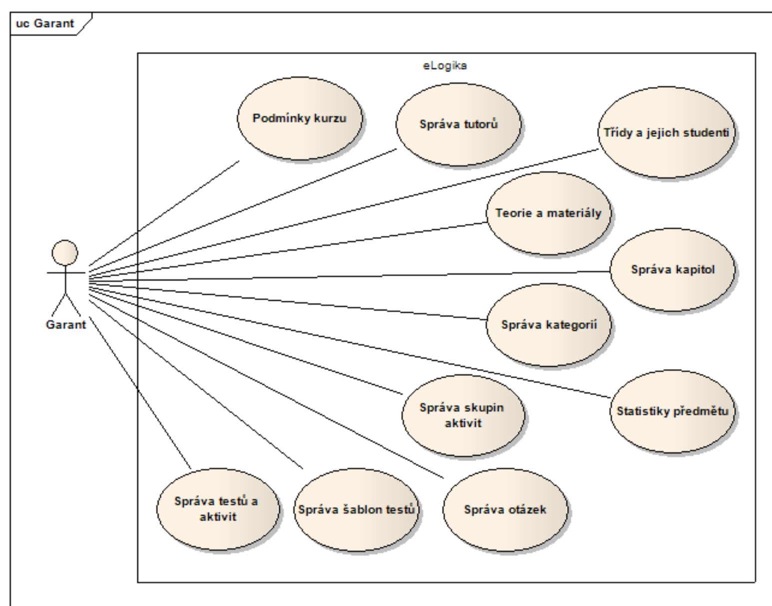
Tajemník je nevyšší autorita v rámci školy. Jeho úkolem je připravit prostředí umožňující výuku pro daný akademický rok. Jde hlavně o vytvoření struktury daného roku, tedy rozdělení na potřebný počet časových úseků, pro které jsou pak definovány kurzy, které se v daném úseku budou vyučovat. Zavedení kurzů a přiřazení garantů zajišťuje rovněž tajemník školy.



Obrázek 2.2 - Diagram případů užití aktéra Tajemník

2.2.3 Role garant

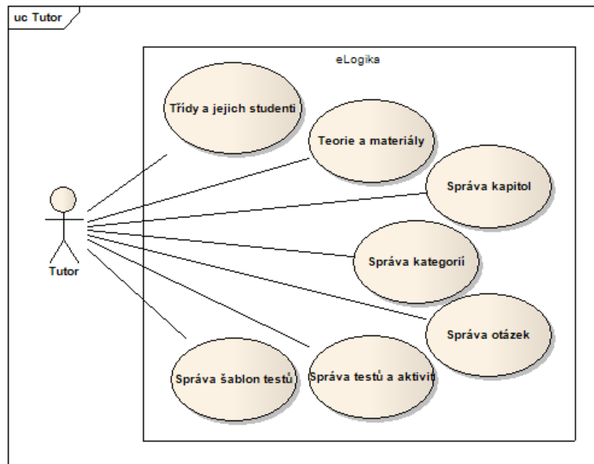
Uživatel v roli garant zodpovídá za celý kurz, kterého se mohou studenti účastnit. Garant určuje náplň kurzu, podmínky pro jeho úspěšné vykonání i strukturu výuky. V jeho režii je většinou i tvorba zkouškových testů, které slouží k závěrečnému ověření znalostí studentů.



Obrázek 2.3 - Diagram případů užití aktéra Garant

2.2.4 Role tutor

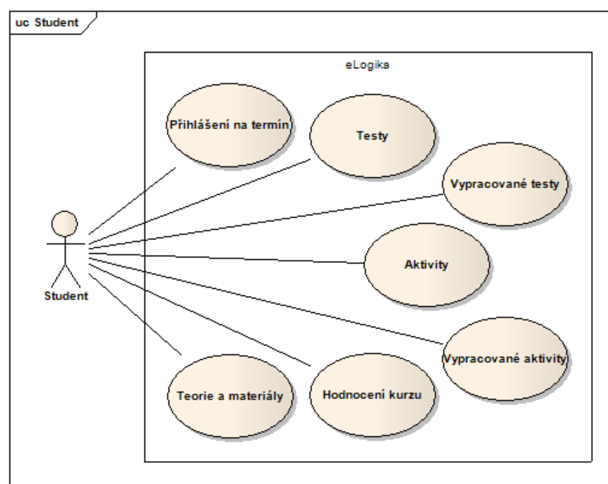
Uživatel v roli tutor by se dal přirovnat k vyučujícímu (cvičícímu) konkrétní skupiny studentů. V rámci eLogiky je tato skupina pojmenovaná jako třída. Hlavní náplní tutora je definování a přidělování úkolů studentům, příprava testů a hodnocení výsledků studentů.



Obrázek 2.4 - Diagram případů užití aktéra Tutor

2.2.5 Role student

Student je nejníže postavená role v rámci hierarchie zodpovědnosti. V rámci celého systému eLogika jde ovšem o stěžejní roli, jelikož uživatelé v roli student jsou ti hlavní, kteří budou se systémem pracovat. Možnosti, které má student v rámci systému, odpovídají činnostem, které student vykonává při běžné výuce. Má možnost nahlížet do studijních materiálů, provádět cvičné testy na danou problematiku, přihlašovat se k vykonání úkolu, nebo se zapisovat na zkoušky.



Obrázek 2.5 - Diagram případů užití aktéra Student

3 Správa přístupu k databázi systému eLogika

Účel všech útoků na webové aplikace je manipulace s daty. Rozdíl je jen v tom, zda je primárním cílem útoku získání, či poškození dat, nebo je manipulace s daty pouhý prostředek, kterým chce útočník dosáhnout svého cíle. V obou případech může řízený přístup k datům útok zkomplikovat, případně mu zcela zabránit.

Aby uživatel aplikace mohl přistupovat k datům, uloženým v databázi, musí se k ní připojit. Parametry tohoto připojení, tedy název serveru, název databáze, login, heslo a další údaje, jsou definovány v přihlašovacím řetězci, tzv. ConnectionStringu (3).

Většina webových aplikací k databázi přistupuje prostřednictvím jediného přihlašovacího jména a hesla. Výhodou tohoto konceptu je jeho implementační jednoduchost, snadná údržba a minimální zatížení jak webové aplikace, tak databázového serveru. S těmito výhodami však přichází značné bezpečnostní riziko. Uživatel databázového serveru má kompletní přístup k celé databázi. Ne zřídka se u malých projektů (kde SQL server využívá jen jedna aplikace) stává, že přihlašovací údaje patří administrátorovi celého serveru. Tento způsob je současně nejnebezpečnější. Pokud dojde k prolomení ochrany webové aplikace, případně zcizení přihlašovacích údajů, má útočník možnost spouštět vlastní kód na serveru. Získá tak plnou kontrolu nad konkrétní databází, potažmo nad celým serverem. Tento přístup také přináší nutnost implementovat veškeré bezpečnostní prvky na straně aplikace. Nejčastěji pak jde o omezení, a současně logování přístupu k funkcím a datům aplikace. Tyto činnosti zvyšují výkonové i časové nároky jak na samotnou aplikaci, tak i databázový server.

Bezpečnější způsob je použít, pro dané připojení, omezenou množinu práv. Pokud jen načítáme data a ty následně vypisujeme do aplikace, stačí přiřadit danému účtu na SQL serveru právo čtení z potřebných tabulek. V případě napadení získá útočník přístup jen k omezené části databáze.

V systému eLogika je přístup k databázi řešen právě tímto způsobem. Kvůli dalším bezpečnostním prvkům je ovšem zpracován ještě detailněji, což bude popsáno v následujících kapitolách.

3.1 Uživatelé aplikace a databázového serveru systému eLogika

Z pohledu práce se systémem eLogika a různého přístupu k databázi, jsem definoval čtyři druhy uživatelů. Jsou jimi běžný uživatel, administrátor, systém a speciální uživatelé.

3.1.1 Běžný uživatel

Jde o jakéhokoliv uživatele[1], který má možnost pracovat se systémem eLogika. Běžný uživatel může v rámci systému vystupovat v jedné z rolí, popsané v kapitole 2.2.

Přístup k databázi je zde řešen na nejnižší možné úrovni. Pro každého uživatele v aplikaci je vytvořen účet na databázovém serveru, pomocí kterého může přistupovat k datům. Pokud se uživatel připojí k aplikaci, jsou pro komunikaci s databázovým serverem použity přihlašovací údaje vytvořené jen pro tohoto konkrétního uživatele. Důvod takovéhoto řešení spočívá v možnosti detailního omezení oprávnění na objekty databáze, pro každého uživatele. Další výhodou je

[1] Vyjma uživatele v roli administrátor

možnost identifikovat konkrétní původce všech požadavků na databázový server, čehož využívám při analýze AQL auditu popsaného v kapitole 0.

3.1.2 Administrátor

V roli administrátor vystupuje v rámci celého systému pouze jeden uživatel. Na rozdíl od běžného uživatele, tedy nelze administrátora vytvořit, ani odstranit, pomocí aplikace. Z toho plyne, že uživatel v roli administrátor je zaveden při inicializaci systému, při které je pro něj vytvořen i účet na databázovém serveru.

3.1.3 Systém

Tento uživatel byl zaveden za účelem zprostředkovat přihlášení k aplikaci běžným uživatelům. Pod tímto uživatelem se nedá do aplikace přihlásit, a tedy ani provádět žádné operace. Fakticky jde pouze o účet na databázovém serveru, který má potřebná práva k tomu, aby uživatele aplikace provedl celým přihlašovacím dialogem.

V procesu autentizace, nemůže aplikace pracovat s přihlašovacími údaji autentizovaného uživatele, avšak potřebuje přístup k databázi. Tento přístup je umožněn právě díky uživateli systém.

3.1.4 Speciální uživatelé

V tomto případě nejde o uživatele aplikace, ale výhradně o uživatele databázového serveru, ke kterému mají navíc znemožněno přihlášení.

Takové uživatele lze použít pouze k tomu, aby se jejich prostřednictvím (v jejich kontextu) provedla některá z nebezpečných operací. Tento, na první pohled zbytečný, mezikrok byl zaveden z důvodů zvýšení bezpečnosti databázového serveru.

Aby mohla být vykonána některá potencionálně nebezpečná operace, musí mít uživatel, který ji chce provést, právo IMPERSONATE, neboli zosobnění, pro konkrétního speciálního uživatele.

3.1.4.1 elogika_userAccess

Tento uživatel slouží pro manipulaci s databázovými uživateli. Konkrétně může vytvářet nové uživatele SQL serveru a databáze, editovat jejich údaje (povolení/zakázaní přihlášení k serveru) a přidělovat/odebírat uživatelům členství v databázových rolích.

Výpis konkrétních oprávnění, která má uživatel elogika_userAccess:

- ALTER ANY LOGIN (na serverové úrovni) – pro manipulaci s přihlašovacími údaji k SQL serveru (vytváření, editace)
- SELECT pro systémové tabulky sql_logins a sysusers – pro možnost ověření listence loginu a uživatele v rámci SQL serveru
- ALTER ANY USER (na databázové úrovni) – pro možnost manipulace s uživateli databáze (vytváření, editace)
- ALTER ANY ROLE – pro možnost přidělování členství v rolích
- ALTER pro každou jednotlivou uživatelskou roli – kvůli možnosti přidělení uživateli členství v dané roli

3.2 Mapování uživatelů systému eLogika na uživatele databáze

Kvůli lepší orientaci v následujícím textu nejprve vysvětlím, jakým způsobem jsou chápány jednotlivé obecné pojmy.

- uživatel databáze – objekt SQL serveru (user) s právy na připojení ke konkrétní databázi
- login – objekt SQL serveru (login), umožňující připojení k SQL serveru
- uživatel – entita požadující přístup k databázi

Přihlášení uživatele k Microsoft SQL Serveru probíhá na dvou úrovních. Nejprve je nutné autentizovat uživatele v rámci celého serveru. Tedy pomocí jeho loginu a hesla ověřit, zda má povolen přístup k serveru. Pokud je uživatel autentizován, může vykonávat úkoly v kontextu databázového serveru (pakliže má potřebná oprávnění). Druhá úroveň zahrnuje připojení ke konkrétní databázi. To už není prováděno na základě hesla, ale jen přiřazení uživatele databáze k danému loginu. Uživatel databáze má pro jednu konkrétní databázi množinu oprávnění, která mu zpřístupňuje potřebnou část databáze.

K jednomu loginu, muže být přiřazeno více uživatelů databáze, přičemž každý z nich je uživatel jiné databáze. Tak je možné, aby uživatel databázového serveru, prostřednictvím jednoho loginu a hesla, pracoval s několika databázemi.

3.2.1 Uložení přihlašovacích údajů k databázi

Z výše uvedeného postupu přihlášení k databázovému serveru je jasné, že aby se mohl konkrétní uživatel, přihlášený k systému eLogika, spojit s databází, musí být login a heslo, které mu bylo vytvořeno, uloženo také na aplikačním serveru. Vzhledem k tomu, že jde o velmi citlivé údaje, musí být uloženy na bezpečném místě. Vhodným úložištěm pro tyto údaje je v rámci webové aplikace, konfigurační soubor web.config. Ten je sice součástí webové aplikace dostupné z internetu, avšak přístup k němu pomocí webového prohlížeče, je odepřen. Problém je, že při změně tohoto souboru dochází k automatickému restartování webové aplikace (z důvodu možné změny nastavení aplikace). Proto se pro účel uchovávání dynamicky generovaných údajů, což přihlašovací údaje nově vytvořených uživatelů bezesporu jsou, nehodí. Restartování aplikace totiž způsobí ztrátu všech neuložených dat a tedy i informací o uživatelích, kteří aktuálně s aplikací pracují.

Jak jsem zmínil v předchozí kapitole, tak v systému eLogika jsou dva uživatelé, jejichž přihlašovací údaje k databázi se nevkládají, ani neupravují za chodu aplikace. Jsou jimi administrátor a systém. Přihlašovací údaje pro tyto uživatele jsou vytvořeny při inicializaci aplikace a tak jejím restartováním nevznikne žádná škoda. Údaje těchto dvou uživatelů jsou tedy uloženy v souboru web.config.

Přihlašovací údaje k databázi ostatních uživatelů, tedy těch, kteří jsou vytvoření prostřednictvím systému eLogika, jsou ukládány do systémového registru Windows, popsaného v (4). K němu má povolen přístup pouze administrátor aplikačního serveru a jde tedy rovněž o způsob bezpečného uložení údajů. Při manipulaci s registry navíc nevzniká žádné omezení pro webovou aplikaci a nic tak nebrání jejich použití.

3.2.1.1 Uložení v souboru web.config

Konfigurační soubor web.config má pro uchování přihlašovacích údajů speciální sekci. Tou je

connectionStrings. Ta obsahuje jednotlivé přihlašovací řetězce, přičemž každý je určen unikátním jménem.

Jeho struktura je následující:

- name – unikátní jméno přihlašovacího řetězce
- connectionString – řetězec obsahující všechny potřebné údaje pro spojení s SQL serverem
- providerName – poskytovatel připojení

Samotný přihlašovací řetězec (connectionString) je uložen v podobě čistého textu. Login k databázovému serveru i heslo jsou tedy čitelné. Pokud by se tedy někdo dostal k samotnému souboru web.config, mohl by uvedené údaje zneužít.

Řešením je zašifrování celé sekce connectionStrings.

- První možností, jak toho docílit, je programové šifrování. Tedy přímo z webové aplikace, například při inicializaci. Sekce získaná z konfiguračního souboru, je následně pomocí metody ProtectSection zašifrována. Tato metoda vyžaduje jako svůj parametr kontejner s šifrovacím klíčem. Výchozí kontejner RsaProtectedConfigurationProvider je vytvořen při instalaci .NET frameworku.

  ```
  protected void Encrypt()
  {
   Configuration cfg = WebConfigurationManager.OpenWebConfiguration
     (Request.ApplicationPath);
   ConnectionStringsSection connSection = cfg.ConnectionStrings;
   if (!connSection.SectionInformation.IsProtected)
   {
    connSection.SectionInformation.ProtectSection("RsaProtected
      ConfigurationProvider");
    cfg.Save();
   }
  }
  ```

- Druhou možností je využít utilitu aspnet_regiis.exe (5), která se nachází na systémovém disku v adresáři windows\Microsoft.NET\Framework\ kde dále vybereme adresář podle verze frameworku. Celá cesta muže vypadat například takto: C:\windows\Microsoft.NET\Framework\v2.0.50727. Parametry, které jsou pro (de)šifrování třeba jsou následující:
  - pe "sekce" – zašifruje zadanou sekci v konfiguračním souboru
  - pd "sekce" – dešifruje zadanou sekci
  - app "/nazev" – udává název aplikace v IIS pro kterou se (de)šifrování provede

  Příkaz:

  ```
  aspnet_regiis.exe -pe "connectionStrings" -app "/eLogika"
  ```

  tak zašifruje sekci connectionStrings webové aplikace, která má v IIS název eLogika.

V obou případech je šifrování transparentní, nijak tedy nenarušuje funkčnost aplikace. V případě žádosti o přihlašovací řetězec, stará se o jeho dešifrování samotná technologie ASP.NET.

3.2.1.2 Uložení v registru Windows

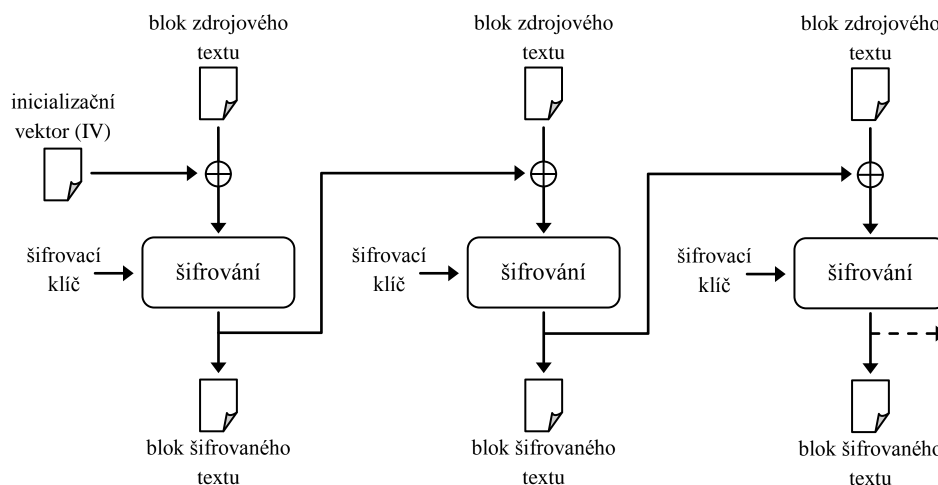
Přihlašovací údaje uživatelů jsou ukládány do klíče *connStr*, který je uložen v klíči *eLogika*. Ten je

uložen jako podklíč klíče *HEKY_LOCAL_MACHINE/SOFTWARE*. Celá cesta v struktuře registru je tedy *HEKY_LOCAL_MACHINE/SOFTWARE/eLogika/connStr*. Údaje pro každého uživatele jsou pak uloženy jako hodnoty tohoto klíče. Název hodnoty je shodný se systémovým loginem (loginem v aplikaci eLogika) uživatele a obsah hodnoty je samotné heslo, použité pro login na databázovém serveru.

I přes to, že by zde měly být hesla v bezpečí, jsou ukládány v zašifrované podobě. Pro šifrování je použit algoritmus AES (Advanced Encryption Standard), který je implementací šifry Rijndael. AES jakýmsi nástupcem za starší algoritmus DES (Data Encryption Standard) a patří do skupiny symetrických algoritmů, používá tedy stejný klíč pro zašifrování i dešifrování. To by mohl být problém při odesílání zašifrovaných dat prostřednictvím sítě. Muselo by se totiž zajistit, že se šifrovací klíč k příjemci dostane bezpečnou cestou. V našem případě se ovšem šifrování i dešifrování provádí na stejném serveru, a tak stačí zajistit bezpečné uložení klíče v rámci daného serveru.

AES je blokový algoritmus, šifruje tedy data s pevně stanovenou délkou, kterou je 128 bitů. Pokud je některý blok kratší (většinou poslední blok šifrovaných dat), je doplněn na požadovanou délku. Pro doplnění bloku se nejčastěji používá postup dle PKCS #7, což je jeden ze standardů pro kryptografii s veřejným klíčem. Pro algoritmus AES, implementovaný v .NET Frameworku, existuje několik způsobů (módů) jak šifrovat jednotlivé bloky. Nejjednodušším je Electronic Codebook (ECB), který zašifruje jednotlivé bloky tak jak jdou za sebou. Tento způsob je však doporučeno nepoužívat, jelikož ve výsledném zašifrovaném řetězci lze rozpoznat stejná schémata jako v původních datech. Další způsoby, které .NET Framework nabízí jsou Cipher Block Chaining (CBC), Cipher Feedback Mode (CFB), Cipher Text Stealing (CTS) a Output Feedback Mode (OFB). Podrobně jsou všechny módy šifrování bloků popsány v (6).

Já zvolil mód CBC. Ten se vyznačuje tím, že každý blok dat je před zašifrováním XORován šifrou předchozího bloku. Nikdy se tak nestane, že by dva stejné bloky původních dat měly stejnou šifru. Aby bylo možné stejným způsobem zašifrovat i první blok dat, který nemá předchůdce, je třeba použít takzvaný inicializační vektor (IV). Inicializační vektor je blok náhodných dat, který je zašifrován běžným způsobem (jakoby s použitím módu ECB) a jeho šifra je pak použita pro zašifrování prvního bloku původních dat. Způsob šifrování jednotlivých bloků je názorně vidět na obrázku Obrázek 3.1 - Šifrování dat metodou CBC. Dešifrování se provádí obdobně. Nejprve je blok šifrovaných dat dešifrován a poté je XORován předchozím blokem šifrovaných dat. Výsledkem je blok zdrojových dat.



Obrázek 3.1 - Šifrování dat metodou CBC

Z výše uvedeného vyplývá, že pro dešifrování zašifrovaných dat potřebujeme znát klíč a inicializační vektor, které byly použity pro zašifrování. Tyto dva údaje je tedy třeba bezpečně uložit na serveru. Vzhledem k tomu, že obě operace se provádějí prostřednictvím webové aplikace, je vhodným místem pro tyto údaje opět soubor web.config. Pro nastavení webové aplikace (za což můžeme klíč a inicializační vektor považovat) je vyhrazena sekce appSettings. Její položky tvoří dvojice key - value, kde key označuje název hodnoty, a value samotnou hodnotu. Celá tato sekce jde, stejně jako sekce connectionStrings, transparentně zašifrovat.

Nyní již k samotnému šifrování databázových hesel. Celý postup se skládá z několika částí. Nejdříve se ze sekce appSettings načtou hodnoty šifrovacího klíče a inicializačního vektoru. Po té se vytvoří nová instance manažeru šifrovacího algoritmu a nastaví se potřebné hodnoty. Vstupní hodnoty jsou typu byte[], proto se musí převést z řetězcové podoby na pole bajtů. Po zašifrování se výsledek převede opět z pole bajtů na řetězec a ten se uloží jako hodnota klíče v registrech (7).

```
Configuration cfg = WebConfigurationManager.OpenWebConfiguration(
    System.Web.HttpContext.Current.Request.ApplicationPath);
AppSettingsSection appSection = cfg.AppSettings;
byte[] keyByte = Convert.FromBase64String(
    appSection.Settings["encryptKey"].Value);
byte[] IVByte = Convert.FromBase64String(
    appSection.Settings["initialValue"].Value);

System.Security.Cryptography.RijndaelManaged aes =
    new System.Security.Cryptography.RijndaelManaged();
aes.Mode = System.Security.Cryptography.CipherMode.CBC;
aes.Padding = System.Security.Cryptography.PaddingMode.PKCS7;
aes.Key = keyByte;
aes.IV = IVByte;

byte[] passByte = Convert.FromBase64String(dbPass);
byte[] sifraByte;
using (System.Security.Cryptography.ICryptoTransform enc =
    aes.CreateEncryptor())
{
    sifraByte = enc.TransformFinalBlock(passByte, 0, passByte.Length);
}

string sifra = Convert.ToBase64String(sifraByte);
RegistryKey rootKey = Registry.LocalMachine.OpenSubKey(
    "Software\\eLogika\\connStr", true);
rootKey.SetValue(user_name, sifra, RegistryValueKind.String);
```

Dešifrování se provádí velmi podobně. Jen nejdříve načteme potřebnou hodnotu z registru a místo Encryptoru vytvoříme Decryptor (aes.CreateDecryptor). Zbytek kódu zůstává shodný.

3.2.1.3 Záloha a obnovení systémového registru Windows

Problém s uložením přihlašovacích údajů k databázi v systémovém registru Windows nastane ve chvíli, kdy bude nutný přesun systému eLogika na jiný aplikační server, případně dojde k přeinstalování celého serveru. Záznamy o uživatelích budou v takovém případě ztraceny a nebude možné provést přihlášení k databázovému serveru.

Aby tato situace nenastala, dochází automaticky k zálohování všech hodnot a podklíčů klíče eLogika. Zálohování je spuštěno vždy po vytvoření nového uživatele systému eLogika. V případě vytváření nových uživatelů pomocí hromadného importu, je záloha registru provedena jen jednou, po vytvoření všech uživatelů. Zálohování provádí metoda exportRegistry(), která ve složce RegFileBackup, jenž je umístěna v aplikační složce App_Data, vytvoří nový soubor s příponou .reg, do kterého zapíše přihlašovací údaje všech uživatelů. Takto vytvořené soubory jsou součástí kódu celé aplikace a při jejím přenosu na nový server nedojde k jejich ztrátě.

Uživatel systému eLogika vystupující v roli administrátor, má možnost provést obnovu hodnot klíče eLogika v systémovém registru Windows. Tu zajišťuje metoda importRegistry(), která v prvním kroku vytvoří celou strukturu klíčů, kam jsou údaje jednotlivých uživatelů ukládány. V dalším kroku pak načte obsah nejnovější zálohy registru a pro každý záznam vytvoří novou hodnotu klíče connStr. V tomto okamžiku mají opět všichni uživatelé plnohodnotný přístup k systému eLogika.

3.2.2 Struktura řetězce connectionString

Jak jsem zmínil dříve, tak cennestionString je řetězec, který obsahuje údaje potřebné pro spojení s databázovým serverem. Množství parametrů, které lze pro spojení nastavit, je opravdu velké a tak uvedu jen ty, které jsou použity pro spojení v systému eLogika.

- Application Name – název aplikace, která žádá o připojení k databázi. Tento parametr slouží převážně pro informativní účely, kdy lze například při ladění výkonu identifikovat původce náročných dotazů na server.
- Data Source – název nebo IP adresa serveru, ke kterému se chceme připojit. Název serveru lze použít ve chvíli, kdy se SQL server nachází ve stejné lokální síti jako aplikace, která se chce k serveru připojit. V opačném případě je třeba použít IP adresu, kde se daný server nachází. Pokud se SQL server nenaslouchá na standardním portu, lze za čárku zadat i port, na který má být zaslána žádost o připojení.
- Initial Catalog – název databáze, ke které se chceme připojit.
- Integrated Security – tento parametr udává, zda se mají pro připojení použít přihlašovací údaje uvedené v connectionStringu (hodnota false), nebo údaje uživatele aktuálně připojeného k systému windows (hodnota true).
- User ID – uživatelské jméno pro připojení k SQL serveru (objekt login SQL serveru).
- Password – heslo pro připojení k SQL serveru.
- Network Library – určuje protokol, který se pro připojení použije. Výchozí hodnota je ‚dbmssocn‘, což značí protokol TCP/IP.

3.2.3 Vytvoření nového uživatele

Jedno z možných rizik pro bezpečnost dat v jakémkoliv systému, jsou vždy uživatelé. Systém jako takový nemusí být nijak napaden a stejně může dojít k úniku dat. Tedy, že data poskytnuta uživateli prostřednictvím systému, jsou jím odcizena a dále je s nimi zacházeno nepovoleným způsobem. Takovému jednání bohužel nelze jednoduše zabránit. Systém eLogika se alespoň snaží omezit možnost vytváření nových účtů, jejichž účel by mohl být právě takovéto jednání.

Vkládat nové uživatele do systému eLogika je proto možné jen z omezeného rozsahu IP adres, definovaném tajemníkem dané školy. Pokud se uživatel nenachází v daném rozsahu, případně není rozsah IP adres definován, není dovoleno vložit uživatele do systému.

Samotné vložení nového uživatele je možné dvěma způsoby. Buďto přímo zadáním údajů o uživateli do systému, nebo hromadným importem studentů do konkrétní třídy. V obou případech je postup vytvoření nového uživatelského účtu následující:

1. Kontrola rozsahu IP adres. Pokud se uživatel nenachází v definovaném rozsahu IP adres pro danou školu, je přesměrován na stránku s potřebnými informacemi.
2. Provede se kontrola, zda se uživatel s danými údaji v systému již nenachází. Pokud ano, načte se id uživatele. Pokud ne, je následujícím způsobem vytvořen nový účet v systému.
    a. Je vygenerován nový systémový login.
    b. Je vygenerováno heslo pro přihlášení do systému, které bude zasláno na uvedený email.
    c. Je vygenerováno heslo pro připojení k databázovému serveru.

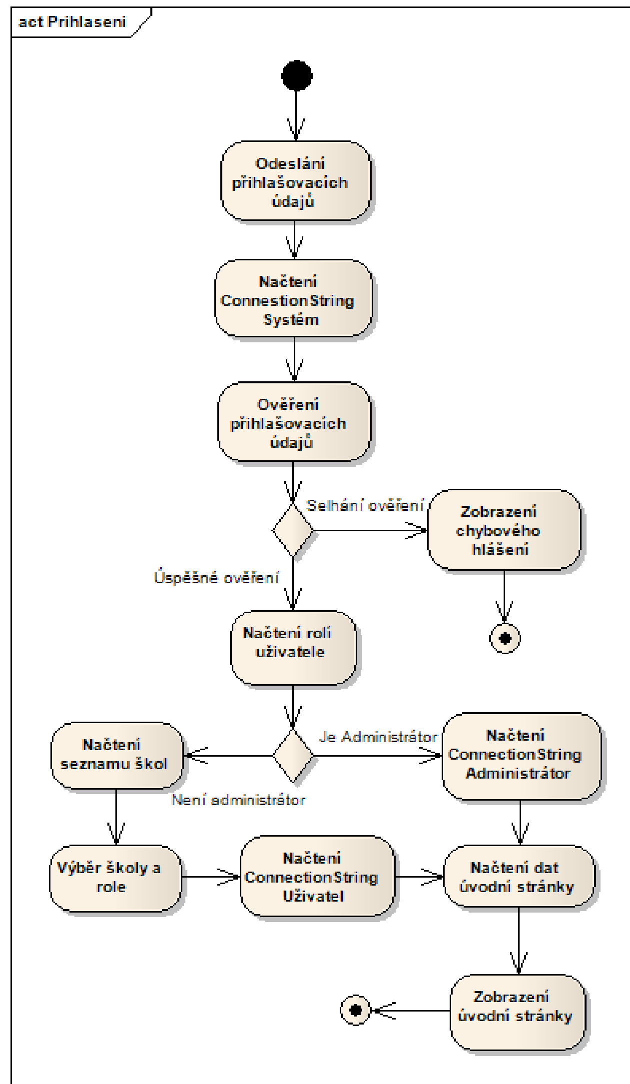
d. Do registru windows jsou uloženy údaje potřebné pro připojení konkrétního uživatele k databázi. [3.2.1.2]
e. Na databázovém serveru jsou vytvořeny nové objekty login a user s názvem shodným jako je vygenerovaný systémový login a s předem vygenerovaným heslem pro připojení k serveru.
f. Do systému je uložen nový uživatel
3. Uživateli je přiřazena požadována systémová role.
4. Je provedena záloha registru Windows s údaji pro připojení uživatelů k databázi.

3.2.4 Přihlášení uživatele do systému eLogika

Jak jsem se zmínil dříve, tak každý uživatel používá pro připojení k SQL serveru vlastní přihlašovací údaje. Ty jsou spjaty se systémovým loginem, který používá uživatel pro přihlášení do eLogiky. Celý proces přihlášení do systému probíhá následovně.

1. Uživatel zadá přihlašovací jméno a heslo.
2. Správce připojení k databázi ověří, zda je uživatel přihlášen. V této chvíli tomu tak není.
3. Správce připojení načte ze souboru web.config connectionString uživatele Systém.
4. Je navázáno spojení s databází pomocí přihlašovacích údajů uživatele Systém.
5. Údaje zadané uživatelem jsou ověřeny s daty v databázi. Pokud nebylo uživatelské jméno nalezeno, případně neodpovídá heslo, je uživateli zobrazeno chybové hlášení.
6. Je načten seznam rolí, kterých je uživatel členem.
7. Pokud je uživatel členem role administrátor, správce spojení načte ze souboru web.config connectionString administrátora.
8. V opačném případě je načten seznam škol, na kterých je uživatel v aktuálním roce členem některé z rolí.
9. Uživateli je zobrazen formulář pro výběr školy a role, do které se chce přihlásit.
10. Uživatel provede výběr a odešle formulář.
11. Správce připojení požádá o connectionString konkrétního uživatele.
12. Pomocí přihlašovacích údajů konkrétního uživatele (případně systémového administrátora) jsou načteny z databáze data, potřebné pro zobrazení úvodní stránky po přihlášení uživatele.

Popsaný proces je znázorněn pomocí diagramu aktivit na obrázku **Chyba! Nenalezen zdroj odkazů.**.



Obrázek 3.2 - Proces přihlášení

3.3 Správa rolí

Pro každého uživatele systému eLogika je nyní vytvořen vlastní účet na SQL serveru, s právy pro připojení k databázi systému eLogika. To ovšem samo o sobě nestačí. Uživatel nemá oprávnění na žádné objekty v databázi, nemá tedy prostředky pro získání dat a manipulaci s nimi.

V nejhorším možném případě by každý uživatel dostal práva vlastníka databáze a měl tak přístup ke všem datům a operacím nad databází. Tím by ale ztratilo smysl přidělování účtů na SQL serveru každému uživateli systému. Výsledný efekt by byl stejný jako při použití jedněch přihlašovacích údajů k SQL serveru, které by používali všichni uživatelé.

Druhou možností by bylo přiřadit každému uživateli jen oprávnění na tu část dat, kterou potřebuje. Při zneužití daného uživatelského účtu by měl útočník k dispozici jen omezenou část databáze a výrazně by se tak snížily možné škody. To je přesně důvod, proč se zabývat nějakou složitější správou přístupu k databázi. I toto řešení má však jeden hlavní nedostatek. Tím jsou přílišné nároky na údržbu aktuálnosti množiny oprávnění. V případě, že by byl uživatel nově přiřazen do některé z rolí, které doposud nebyl členem, musely by se k jeho databázovému účtu přiřadit oprávnění na

nově dostupné data a funkce. Stejně tak, při odebrání členství v roli, by se musely odebrat oprávnění, které vyplývají ze členství v dané roli. Jednotlivé role však mohou mít část oprávnění shodnou a muselo by se ošetřit i to, že nebudou odebrána oprávnění ve chvíli, kdy uživatel zůstává členem jiné role, která má na dané oprávnění taky nárok.

Nejlepším řešením je sdružovat uživatele, respektive jejich účty na SQL serveru, do skupin podle toho, jaké role zastávají v systému eLogika. Každá z těchto skupin by odpovídala jedné roli v eLogice a obsahovala by oprávnění ke všem datům, potřebným pro danou roli. Pokud by byla uživateli eLogiky přidána nová role, byl by jeho databázový účet přirazen do dané skupiny uživatelů. Tím by měl k dispozici potřebná oprávnění jak v rámci eLogiky, tak jím odpovídající databázové oprávnění. V případě odebrání z některé skupiny, zůstane uživateli množina oprávnění, kterou obsahují ostatní skupiny, ve kterých zůstal.

Takovéto řešení přináší i velké zjednodušení při případných úpravách systému. Pokud bude doprogramována nějaká nová funkce, která pracuje s databází, stačí přidat potřebné oprávnění těm skupinám na SQL serveru, které korespondují s rolemi v eLogice. Potřebné oprávnění bude automaticky přiděleno všem uživatelům v dané roli.

3.3.1 Technologie pro seskupování uživatelů

Microsoft SQL Server 2008 nabízí několik možností, jak seskupovat uživatele a hromadně spravovat jejich oprávnění. Tyto skupiny se v rámci SQL serveru nazývají role a dají se rozdělit do tří skupin.

3.3.1.1 Výchozí serverové a databázové role

SQL server má na serverové, i databázové úrovni připraveno několik rolí, jejichž členem se může uživatel stát. Každá taková role se váže k některé konkrétní činnosti, kterou lze v rámci serveru či databáze provádět. Mezi ně patří například vytváření databází, uživatelů, správa záloh nebo objektů spjatých s bezpečností. Tyto role mají pevně přiřazenu množinu oprávnění, kterou není možné upravovat. Nelze tak omezit přístup jen k některým z množiny objektů a tím pádem se nehodí pro detailní řízení přístupu k objektům databáze.

3.3.1.2 Uživatelské databázové role

Tyto role, jak už název napovídá, jsou definované uživatelem (správcem SQL serveru) a nemají tedy předem určenu množinu oprávnění. Role jsou vytvářeny vždy jen pro jednu konkrétní databázi. Členem databázové role může být uživatel databáze (objekt User), nebo další databázová role. Lze tedy vytvářet jakousi hierarchii rolí, která muže v některých případech značně usnadnit správu oprávnění.

Tím se dostávám k největší výhodě databázových rolí, a tou je možnost přidělit konkrétní roli konkrétní oprávnění. Role může mít různá oprávnění na jakékoliv z celkových 22 typů objektů, mezi které patří například databáze, tabulky, funkce, uživatelé nebo i uživateli definované datové typy. Každý typ objektu pak nabízí sadu jednotlivých oprávnění, které lze dané roli přidělit, odebrat, nebo přidělit s možností dále dané oprávnění přidělovat. Tedy že může svoje oprávnění přidělit jiným rolím nebo uživatelům.

3.3.1.3 Aplikační role

Aplikačním rolím lze, stejně jako rolím databázovým, přiřadit množinu oprávnění. Tím ovšem podobnost končí. Aplikační role nemá žádné členy a ve výchozím stavu je neaktivní. Aby uživatel získal oprávnění, které jsou nastaveny pro aplikační roli, musí se po přihlášení k SQL serveru identifikovat pomocí hesla, které dané aplikační roli náleží. Celý proces přihlášení k databázi je následující:

1. Uživatel se pomocí svého loginu a hesla připojí k SQL serveru.
2. Uživatel, nebo spíše aplikace, spustí systémovou proceduru SQL serveru sp_setapprole. Její parametry jsou název aplikační role a heslo pro její aktivaci.
3. V tomto okamžiku spojení s SQL serverem ztrácí oprávnění, které byly přiděleny uživateli, který spojení navazoval, a získává všechna oprávnění, které má přiřazena daná aplikační role.

Hlavní myšlenka aplikačních rolí spočívá v omezení „míst“ ze kterých se lze k databázi připojit. Pokud bychom třeba měli dvě různé aplikace, které pracují se stejnou databází, přiřadíme každé jednu aplikační roli s heslem a dané aplikace upravíme tak, aby po navázání spojení s SQL serverem spustily uloženou proceduru sp_setapprole. Nebude-li mít k databázi přístup jiná role, nebo uživatel, dosáhneme toho, že se k databázi dá připojit jen pomocí oněch dvou aplikací.

Dalo by se namítnout, že stejného výsledku lze dosáhnout i pomocí uživatelských databázových rolí. U těch ovšem útočníkovi stačí získat přihlašovací jméno a heslo a má k dispozici alespoň část databáze. U aplikačních rolí to nestačí. Po přihlášení k SQL serveru by útočník neměl žádné oprávnění. Ty by získal až ve chvíli, kdy by se mu podařilo zcizit i jméno a heslo k aplikační roli. Tyto údaje by však měly v aplikaci být uchovány tak, aby je nebylo možné snadno získat.

Dá se tedy říct, že aplikační role vkládají do procesu přístupu k datům v databázi, další bezpečnostní krok. Rovněž ještě více zjednodušují správu oprávnění na databázové úrovni. Není třeba přiřazovat či odebírat uživatelům databáze členství v rolích. O výběr role se stará již aplikace, která určí jaká role se má použít.

Zdálo by se, že použití aplikačních rolí pro správu oprávnění, je pro systém eLogika nejlepší z uvedených možností. Bohužel je nelze použít z důvodu, který jsem zmínil dříve. Po té, co je aktivována aplikační role, ztrácí se pro toto připojení všechna oprávnění přidělená loginu / uživateli, který spojení vytvořil. Některé operace prováděné v databázi ovšem potřebují oprávnění IMPERSONATE na speciální uživatele databáze uvedené v kapitole 3.1.4. Toto oprávnění je přidělováno konkrétnímu loginu SQL serveru a nelze jej přidělit aplikační roli.

Vzhledem k tomu, že aplikační role nelze pro mnou navrženou správu oprávnění použít, rozhodl jsem se pracovat s uživatelskými databázovými rolemi.

3.3.2 Oprávnění rolí

Každá role v systému eLogika (role jsou popsány v kapitole 2.2), má povolen přístup jen k některým částem celé aplikace, a v rámci těchto částí k některým operacím. Jelikož jde o webovou aplikaci, tak části lze chápat jako jednotlivé webové stránky, a operace jako akce, které na těchto stránkách lze provést. Na stránce se seznamem uživatelů jsou tedy akce jako zobrazení profilu uživatele, editace profilu uživatele nebo smazání uživatele.

Přidělování oprávnění na stránky a akce, a jejich kontrolu, obstarává samotná aplikace. Pokud přijde požadavek na zobrazení seznamu uživatelů, samotná aplikace ověří, zda role, pod kterou uživatel vystupuje, má oprávnění na zobrazení stránky s výpisem uživatelů. Pokud ano, načte aplikace z databáze seznam uživatelů a sestaví tabulku pro jejich vypsání do stránky. Současně ověří oprávnění na všechny akce, které stránka poskytuje. Podle jednotlivých oprávnění pak do stránky vykreslí potřebné ovládací prvky. Tím je zajištěna bezpečnost dat při práci prostřednictvím aplikace. Každý uživatel má přístup jen k těm datům, ke kterým mu to umožňuje role, pod kterou vystupuje. Aby stejná úroveň zabezpečení dat zůstala zachována i ve chvíli, kdy se útočník přihlásí přímo k databázi, je třeba přístup k datům omezit i na databázové úrovni.

3.3.2.1 Vytvoření databázové role

K tomuto omezení jsem využil uživatelské databázové role, jejichž výhody jsem zmínil v kapitole 3.3.1.2. Pro každou roli v systému eLogika, je vytvořena databázová role na SQL serveru, jejíž název je odvozen od názvu role v aplikaci. Z důvodu jedinečnosti názvu v rámci databáze, jsou názvy rolí rozšířeny o prefix *elogika_* a sufix *_R*. O vytvoření rolí se na SQL serveru stará systémová procedura *sp_addrole*, která jako parametr očekává název role. Role je vytvořena v databázi, v jejímž kontextu je procedura zavolána. Příklad na dotaz vytvářející novou databázovou roli:

```
USE [eLogika]
GO
EXEC sp_addrole 'elogika_system_R'
GO
```

3.3.2.2 Přidělení oprávnění roli

Každý uživatelův požadavek na zobrazení stránky, nebo provedení akce, vyvolá řadu požadavků na data (či operace s daty), které zasílá aplikace databázovému serveru. Jelikož množina všech stránek a akcí, na které má konkrétní role práva v rámci aplikace, je známa, lze definovat množinu dat a operací s nimi, které daná role vyžaduje.

Nyní by bylo možné, přidělit databázovým rolím oprávnění na operace SELECT, INSERT, UPDATE a DELETE pro tabulky, se kterými musí pracovat. Výsledný stav by ovšem byl takový, že by každá role pracovala s většinou tabulek. Bylo by totiž nutné přidělit oprávnění i na vazební tabulky, jejichž obsah je uživateli skryt, ale jsou potřeba pro vytvoření množin dat, které chceme uživateli zobrazit.

Systém eLogika naštěstí provádí převážnou většinu požadavků na databázi jako volání uložených procedur. Pokud tedy aplikace požaduje seznam všech uživatelů, není výsledný SQL dotaz v klasickém výběrovém tvaru: `SELECT * FROM uzivatele`, ale ve tvaru volání procedury: `EXEC uzivatele_get`, kde tělo procedury uzivatele_get se postará o načtení záznamů z potřebných tabulek a vrácení výsledku. Pro spuštění procedury je zapotřebí pouze oprávnění EXECUTE a nezáleží dále na tom, zda se v těle procedury používá operace SELECT nebo DELETE, ani na množství použitých tabulek. Použití procedur přináší také další zvýšení bezpečnosti. K datům nelze přistupovat přímo a bez přidělených oprávnění na tabulky nelze ani zjistit jejich přesné jména či atributy. Výjimku tvoří několik tabulek, na které je přiděleno oprávnění SELECT. Tento přístup vyžaduje objekt TableAdapter, který je použit pro načítání některých dat.

Aby mohla role oprávnění poskytnout uživatelům, musí jí být nejprve přiděleny. To je provedeno pomocí příkazu GRANT, po kterém následuje typ oprávnění, objekt na který se oprávnění vztahuje a objekt kterému oprávnění přidělujeme.

```
GRANT SELECT
      ON [skola]
      TO "elogika_system_R"
GRANT EXECUTE
      ON [skola_get]
      TO "elogika_system_R"
```

Nyní přichází na řadu nejdůležitější část. Samotné přidělování a odebírání členství v rolích. Pokud by selhalo přidělení členství v roli, tak by se mohlo stát, že se uživatel do aplikace přihlásí v roli, jejíž oprávnění nemá přidělena na databázové úrovni. Byla by mu tedy znemožněna jakákoliv práce v rámci aplikace. V opačném případě, kdy by uživateli nebyla odebrána databázová role, měl by uživatel oprávnění na spuštění procedur, které jeho zbývající role nevyžadují. Vzniklo by tak potenciálně nebezpečné místo, kudy se útočník může dostat k datům[2]. Abych předešel oběma nežádoucím situacím, bylo nutné nadefinovat přesné postupy, jak spravovat členství v rolích.

3.3.3 Přiřazení uživatele k roli

Členství v databázové roli je potřeba přidělit vždy, když je uživateli aplikace přidělena některá role systému eLogika. Tedy při přidání tajemníka, garanta, tutora nebo studenta. Ve všech těchto případech by byl postup přidělení členství v databázové roli stejný, proto jsem danou funkci přesunul do univerzální procedury s názvem elogika_addRole. Ta je pak volána z procedur, které se starají o přidání členství v roli v rámci aplikace.

Procedura elogika_addRole očekává dva vstupní parametry. Prvním z nich je jméno uživatele databáze. To je stejné jako přihlašovací jméno k aplikaci a tak je možné, předat jej jako parametr přímo z aplikace při volání procedury. Druhým parametrem je id role, do níž chceme přidělit uživateli členství.

Samotné přidělení členství zajišťuje systémová procedura sp_addrolemember, která jako jeden z parametrů očekává název databázové role. Ten není aplikaci znám a tak ho musíme určit podle id role. V případě, že přidělujeme uživateli členství v roli, která může dále vkládat uživatele do rolí (což jsou všechny role kromě studenta), musíme loginu SQL serveru daného uživatele, přidělit oprávnění IMPERSONATE, na uživatele databáze elogika_userAccess. Účel tohoto uživatele je popsán v kapitole 3.1.4.

Obě výše zmíněné operace pracují s objekty, na které je potřeba speciální oprávnění. Toto oprávnění není přiděleno rolím, stejně jako oprávnění na přístup k datům, ale speciálnímu uživateli. Proto se tyto operace musí provést v kontextu tohoto uživatele.

```
EXECUTE AS LOGIN = 'elogika_usersAccess_L' EXEC
dbo.sp_addrolemember @rolename=@roleName, @membername=@login
   IF @grantRole = 1
      EXEC('USE [master]; GRANT IMPERSONATE ON
           LOGIN::elogika_usersAccess_L TO ' + @login)
REVERT
```

[2] Za předpokladu zneužití konkrétního uživatelského účtu.

Zjednodušeně tedy platí, že pro přidání uživatele do role, musí mít uživatel, který to chce provést, právo vydávat se za speciálního uživatele elogika_userAccess. V případě, že role, do které je uživatel vkládán, má právo přidělovat role, musíme danému uživateli přidělit i právo vydávat se za speciálního uživatele elogika_userAccess.

Po té, co je uživateli přiděleno členství v roli, má k dispozici všechna oprávnění, které náleží dané roli. Dříve jsem se zmínil, že uživatel může být členem jedné role vícekrát. To platí jen pro role v rámci aplikace. Pokud bude uživateli vícekrát přiděleno členství v databázové roli, bude členem stále pouze jednou. Každé další přidání do stejné role nebude mít tedy žádný vliv.

3.3.4 Odebrání uživatele z role

Odebírání členství z databázové role probíhá do jisté chvíle stejně jako přidělování členství. Z procedur, které zajišťují odebrání role na úrovni aplikace, je zavolána procedura elogika_removeRole, která má opět parametry login a id role. Login je předán z aplikace a dle id role určíme název role, kterou chceme odebrat. Důležité je, že procedura elogika_removeRole je volána až po té, co je uživateli odebrána role na úrovni aplikace.

Pokud bychom nyní odebrali členství v dané roli, mohl by nastat problém ve chvíli, kdy je uživatel členem dané role vícekrát. Přišel by o potřebná práva a v aplikaci by pod danou roli nemohl provádět žádné operace. Je tedy třeba ověřit, zda je uživatel členem dané role na úrovni aplikace. Pokud byl uživatel členem role jen jednou, byla mu z aplikační úrovně odebrána a nyní již není jejím členem. Můžeme tedy odebrat i roli na databázové úrovni. Pokud by ale byl členem role vícekrát, tak by byl jejím členem i po té, co by mu byla role na aplikační úrovni jednou odebrána. V této chvíli mu nelze odebrat roli na databázové úrovni.

O odebrání databázové role se stará systémová procedura sp_droprolemember, která očekává jako parametry název role pro odebrání a login uživatele, kterému roli odebrat. Tuto proceduru je opět nutné spustit v kontextu speciálního uživatele elogika_userAccess.

```
DECLARE @roleCount INT
SELECT @roleCount = COUNT(*) FROM (
   SELECT ur.id_role FROM uzivatel_role ur, uzivatel u
      WHERE ur.id_uzivatel = u.id_uzivatel AND
      u.login = @login AND ur.id_role = @id_role
) AS x
IF @roleCount = 0
BEGIN
   EXECUTE AS LOGIN = 'elogika_usersAccess_L'
      EXEC dbo.sp_droprolemember @rolename=@roleName,
         @membername=@login
   REVERT
END
```

Ještě složitější je situace s odebráním oprávnění, umožňující se vydávat za uživatele elogika_userAccess. Toto oprávnění je totiž přiděleno všem uživatelům, kteří jsou členem jakékoliv role mimo studenta. Po odebrání role na aplikační úrovni je tedy třeba ověřit, zda je daný uživatel členem jakékoliv ze zbývajících tří rolí (na aplikační úrovni). Pokud již není členem žádné takové role, je uživateli odebráno oprávnění IMPERSONATE na uživatele elogika_aserAccess.

```
DECLARE @roleCountAll INT
SELECT @roleCountAll = COUNT(*) FROM (
   SELECT ur.id_role FROM uzivatel_role ur, uzivatel u
      WHERE ur.id_uzivatel = u.id_uzivatel AND
      u.login = @login AND ur.id_role != 5
) AS y
IF @roleCountAll = 0
BEGIN
   EXECUTE AS LOGIN = 'elogika_usersAccess_L'
      EXEC('USE [master]; DENY IMPERSONATE ON
         LOGIN::elogika_usersAccess_L TO ' + @login)
   REVERT
END
```

3.4 Použití koncových bodů[3]

Pokud se chceme vzdáleně připojit k libovolné aplikaci, musíme určit IP adresu a ve většině případů i port (případně v rámci lokální sítě jinak identifikovat server, kde aplikace běží), na kterém aplikace očekává požadavek na spojení. Jde tedy o jakási vstupní místa aplikace.

K Microsoft SQL Serveru lze přistupovat pomocí čtyř různých protokolů. Jsou to Shared Memory, Named Pipes, TCP/IP a VIA. Vzhledem k tomu, že s SQL serverem chceme pracovat prostřednictvím webové aplikace, budeme potřebovat pouze protokol TCP/IP. V případě, že bude potřeba k nastavení SQL serveru využít SQL Server Management Studio, tak použijeme i protokol Named Pipes. Zbylé dva protokoly je možné zakázat. To provedeme pomocí nástroje SQL Server Configuration Manager. V sekci SQL Server Network Configuration vybereme naší instanci serveru (defaultně MSSQLSERVER) a v kontextové nabídce jednotlivých protokolů zvolíme Disabled.

Pro každý z uvedených protokolů lze vytvořit takzvaný Endpoint, čili koncový bod (8). Ty slouží tomu, aby bylo možné konkrétním uživatelům SQL serveru určit, pomocí kterého protokolu se smí k serveru připojit. Lze tak dosáhnout toho, že někteří uživatele se mohou připojit jen pomocí protokolu TCP/IP a portu 12345, jíní pomocí portu 54321 a ostatní jen pomocí protokolu Named Pipes.

Po vytvoření nové instance SQL serveru jsou vytvořeny výchozí koncové body a tak je umožněno spojení pomocí všech výše uvedených protokolů pro všechny členy role public. Což jsou všichni uživatele SQL serveru. Výchozí hodnoty koncového bodu pro protokol TCP/IP jsou nastaveny tak, že jsou akceptována všechna připojení, které vyhoví omezení protokolu TCP/IP.

3.4.1 Omezení přístupu k SQL serveru

Aby byl ztížen případný útok na SQL server, je vhodné změnit výchozí port na některý jiný volný, z rozsahu 1024-65535. Pokud by útočník pro spojení se serverem použil IP adresu, na které je server dostupný, ale neuvedl by port, pokus o připojení by selhal. V případě, že není zadán port, zkusí se totiž použít port výchozí. Na něm ovšem SQL server, po změně portu, neočekává požadavky na spojení.

Změna portu se provede pomocí nástroje SQL Server Configuration Manager. V sekci *SQL Server*

---

[3] Postup nastavení koncových bodů byl převzat z (12 stránky 252-258).

Network Configuration vybereme naší instanci serveru (defaultně MSSQLSERVER) a zobrazíme si vlastnosti protokolu TCP/IP. Na kartě *Protocol* nastavíme položku *Listen All* na hodnotu No. Můžeme tedy určit, na které IP adrese budou požadavky o spojení akceptovány. To provedeme na kartě *IP Addresses*, kde jsou pole pro určení dvou IP adres. Jedna z nich se většinou používá pro přístup z internetu, druhá pro localhost, či vnitřní síť. U jedné adresy tedy nastavíme pole *Active* i *Enabled* na No a druhou IP adresu nastavíme na 127.0.0.1 s portem 6000. Po změně nastavení je třeba restartovat SQL server.

Nyní bude SQL server přijímat pouze požadavky o spojení posílané na adresu 127.0.0.1 a port 6000. ConnectionString pro navázání spojení by nyní musel vypadat takto:

```
Data Source=127.0.0.1,6000;Network Library=dbmssocn;Initial
Catalog=nazev_databaze;User ID=login_uzivatele;Password=heslo_uzivatele
```

Dalo by se říct, že tato úprava nemá s koncovými body nic společného. Ovšem je to jen proto, že výchozí koncový bod pro protokol TPC/IP akceptuje všechna připojení. Tedy i na adrese 127.0.0.1 a portu 6000.

3.4.2 Vlastní koncové body

V některých případech může být omezení uživatelů, na možnost připojení k SQL serveru jen pomocí daného protokolu s danými parametry, dost svazující, nebo i ke škodě. V případě webové aplikace tomu tak není. Požadavky na SQL server posílá vždy daná aplikace, potažmo aplikační server. Druh komunikace mezi aplikačním a SQL serverem se jednou nastaví a nadále se nemění, proto není konkrétní, pevně určený protokol, IP adresa či port žádným omezením.

Naopak s takovýmto řešením přichází další bezpečnostní opatření. Pokud by útočník získal přihlašovací údaje některého uživatele, musel by pro připojení použít parametry, které akceptuje koncový, bod pomocí kterého se daný uživatel připojuje. Což může znamenat, že se útočník musí nacházet v lokální síti kde je umístěn server, nebo že musí mít přístup přímo k danému serveru.

Nový koncový bod vytvoříme pomocí uvedeného SQL dotazu.

```
CREATE ENDPOINT [eLogikaEndpoint]
STATE = STARTED
AS TCP
    (LISTENER_PORT = 6000
    ,LISTENER_IP = (127.0.0.1))
FOR TSQL () ;
GO
```

Po vytvoření nového koncového bodu se automaticky odebere oprávnění CONNECT k výchozímu koncovému bodu pro roli public. To způsobí, že všem uživatelům, kteří se přes daný protokol pokusí připojit, navazování spojení selže. Aby se uživatelé mohli připojit, musíme jim přidělit oprávnění CONNECT pro nově vytvořený koncový bod.

```
GRANT CONNECT ON ENDPOINT::[eLogikaEndpoint] TO [login_uzivatele]
```

Bohužel nelze oprávnění na koncový bod přidělit databázové či aplikační roli. Lze je přidělit pouze serverové roli nebo loginu SQL serveru. Není ovšem problém zahrnout přidělení oprávnění na koncový bod do procesu vytváření nového uživatelského účtu. Tím se zajistí, že všichni uživatelé vytvoření pomocí systému eLogika, budou mít všechna potřebná oprávnění pro spojení s SQL serverem.

Pokud by bylo potřeba obnovit přístup k serveru pro všechny uživatele, stačí znovu přidělit práva na výchozí koncový bod serverové roli public.

```
GRANT CONNECT ON ENDPOINT::[TSQL DEFAULT TCP] TO [public]
```

3.5 Výchozí uživatelé SQL Serveru

3.5.1 Super Administrátor (sa)

Super administrátor je uživatel s neomezenými právy v rámci celého serveru, tedy i všech databází. Proto by se bezpečnost tohoto účtu neměla zanedbávat. Základem by vždy mělo být jeho přejmenování, aby útočník při pokusu o jeho zneužití, neměl práci usnadněnu výchozím pojmenováním.

```
ALTER LOGIN [sa] WITH NAME = [superadmin]
```

Hned druhým krokem by mělo být nastavení silného hesla. Ve výchozím nastavení totiž nemá uživatel sa přiřazeno žádné heslo. Nové heslo by mělo být dostatečně dlouhé, mělo obsahovat malé i velké písmena, číslice i speciální znaky.

```
ALTER LOGIN [superadmin] WITH PASSWORD = 'dostatecneSilneHeslo'
```

Nejbezpečnější je účet super administrátora úplně zakázat. To je možné ve chvíli, kdy není třeba SQL server nastavovat vzdáleně a je na serveru povolen Windows Authentication mód pro přihlášení k serveru. Pomocí nástroje SQL Server Management Studio tak je možné kdykoliv účet sa opět povolit.

```
-- zakazani uctu
ALTER LOGIN [superadmin] DISABLE
-- povoleni uctu
ALTER LOGIN [superadmin] ENABLE
```

3.5.2 Guest

Jde o speciální databázový účet, který se nachází u všech databází. A to jak vytvořených uživatelem, tak i u systémových databází. Účet Guest umožňuje přístup k databázi bez uživatelského účtu spjatého s loginem SQL serveru.

Přístup k databázi pomocí účtu Guest je povolen při splnění těchto podmínek:

1. Uživatel má povoleno přihlášení k SQL serveru, ale nemá účet pro danou databázi.
2. Databáze, ke které se chce uživatel připojit, obsahuje účet Guest.
3. Účet Guest má v databázi přiděleno oprávnění CONNECT.

Vzhledem k tomu, že účet Guest je v databázi od jejího vytvoření, nemá ve výchozím stavu žádná oprávnění na žádné objekty vytvořené uživatelem. Nelze tedy prostřednictvím tohoto účtu přímo získat data z databáze. Účet Guest má však některá oprávnění na systémové objekty v rámci databáze. Může si tedy volně prohlížet systémové pohledy (views) a z nich získat přehled o obsahu databáze. To je značné bezpečnostní riziko, které však lze snadno odstranit.

V naprosté většině případů je účet Guest v databázi zbytečný a tak není problém znemožnit tomuto účtu připojení k databázi. To se provede odebráním práva connect.

```
USE [eLogika]
GO
REVOKE CONNECT FROM guest
GO
```

U systémových tabulek master a tempdb tento příkaz nelze provést, jelikož používají účet Guest jako prostředníka pro přistupování ke svým datům.

3.6 Zálohování databáze

Nedílnou součástí zabezpečení databáze je její zálohování pro případ poškození či ztráty dat vinou napadení databázového serveru. Pokud by webová aplikace navštěvovaná tisícovkami uživatelů neměla vyřešeno zálohování databází, mohlo by to pro ni mít velmi negativní následky, pokud by uživatelé z nějakého důvodu přišli o svá data. Aby k tomu nedošlo, je třeba navrhnout optimální řešení pro správu záloh.

3.6.1 Typy záloh

Microsoft SQL Server 2008 nabízí čtyři typy záloh. Pro všechny z nich jsou společné některé parametry, které ovlivňují průběh a výsledek zálohování. Mezi ty hlavní patří:

- INIT/NOINIT – Tento parametr určuje, zda bude soubor, do kterého se provádí záloha, přepsán, nebo bude záloha připojena na konec daného souboru.
- CHECKSUM/NO_CHECKSUM – Nastavení, zda se má provést kontrola jednotlivých stránek zálohy a nakonec také kontrola celé zálohy. Vždy je lepší kontrolu povolit. Je lepší zjistit chybu v záloze při jejím vytváření, než ve chvíli, kdy z ní nejde obnovit databáze.
- STOP_ON_ERROR/CONTINUE_PAST_ERROR – V závislosti na předchozím parametru se při nalezení chyby buďto zálohování přeruší, nebo bude pokračovat.
- COMPRESSION/NO_COMRESSION – Verze Enterprise, Microsoft SQL Serveru 2008, umožňuje ukládat zálohy komprimované na jednu čtvrtinu až jednu desetinu původní velikosti. Tato volba sice zvyšuje nároky na vytvoření zálohy, avšak při dlouhodobějším uchovávání záloh, případně při velkém objemu dat je jejich komprese velkou výhodou.

3.6.1.1 Úplná záloha

Jak už název napovídá, tak úplná záloha databáze obsahuje všechny data, které jsou v daném okamžiku v databázi dostupné. V průběhu vytváření zálohy je možné s databází dále pracovat a tedy i měnit data, která jsou zálohována. Aby byla záloha opravdu aktuální, postupuje SQL Server následujícím způsobem (9 str. 308):

1. Dojde k uzamčení databáze a zablokují se všechny transakce.
2. Do transakčního protokolu[4] je umístěna první značka.
3. Uvolnění zámku databáze.
4. Uložení všech dat dostupných v databázi do souboru zálohy.
5. Opětovné uzamčení databáze a zablokování transakcí.
6. Do transakčního protokolu je umístěna druhá značka.
7. Uvolnění zámku databáze.

[4] Transakční protokol je soubor, kam se ukládají všechny změny prováděné v databázi

8. Je extrahována část transakčního protokolu mezi první a druhou značkou a následně je připojena k souboru zálohy.

Úplná záloha je vyžadována jako základ pro rozdílovou zálohu a zálohu transakčního logu.

Ukázka příkazu pro vytvoření úplné zálohy databáze eLogika, s použitím komprese a ověření kontrolních součtů[5]:

```
BACKUP DATABASE eLogika
TO DISK = 'eLogika_1.bak'
WITH COMPRESSION, INIT, CHECKSUM, STOP_ON_ERROR
```

Pokud není uvedena cesta, kam má být soubor zálohy uložen, je použit výchozí adresář pro zálohy dané instance SQL serveru.

3.6.1.2 Rozdílová záloha

Jak jsem se zmínil výše, tak tento typ zálohy lze použít jen ve chvíli, kdy je vytvořena úplná záloha databáze. Důvod je ten, že rozdílová záloha uchovává pouze změny, které se odehrály od poslední úplné zálohy. Oproti opakovanému provádění úplných záloh tak dochází k výrazné úspoře místa. Je třeba zdůraznit, že rozdílová záloha je vždy provedena vůči poslední úplné záloze a ne vůči předchozí rozdílové záloze. Rozdíl v syntaxi oproti úplné záloze je jen v klíčovém slově DIFFERENTIAL, uvedeném v seznamu podmínek.

```
BACKUP DATABASE eLogika
TO DISK = 'eLogika_1.dif'
WITH DIFFERENTIAL, COMPRESSION, INIT, CHECKSUM, STOP_ON_ERROR
```

3.6.1.3 Záloha transakčního protokolu

Stejně jako předchozí typ zálohy, i tento je závislý na vytvořené úplné záloze databáze. Důvod je ten, že transakční protokol neuchovává všechna data, ale pouze jejich změny. Jednotlivé zálohy transakčního protokolu lze navazovat na sebe, kdy každá další obsahuje jen změny oproti předchozí záloze stejného typu (vyjma první zálohy transakčního protokolu, která navazuje na úplnou zálohu). Tím se ještě více sníží množství dat, které se po dobu zálohování ukládají. Důležité je, že jednotlivé zálohy musí být obnovovány jedna po druhé. Dojde li k poškození některé ze záloh, nelze použít jí následující zálohy a dojde tak ke ztrátě značného množství dat.

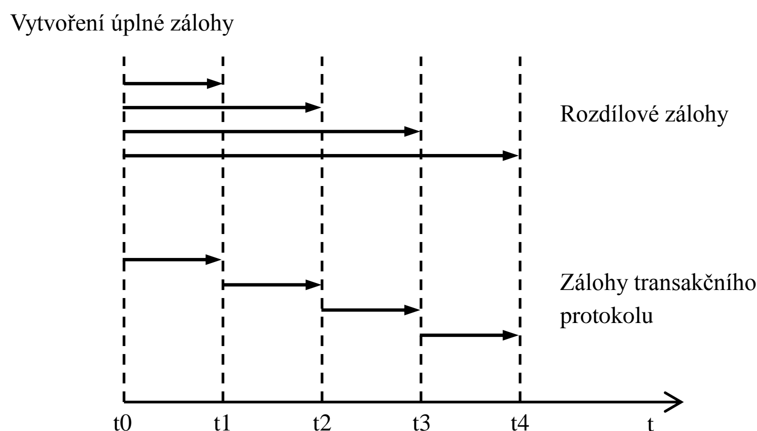
Rozdíl v zápisu příkazu pro vytvoření zálohy, je v klíčovém slově LOG, které nahradilo DATABASE.

```
BACKUP LOG eLogika
TO DISK = 'eLogika_1.trn'
WITH COMPRESSION, INIT, CHECKSUM, STOP_ON_ERROR
```

Rozdíl mezi zálohou transakčního protokolu a rozdílovou zálohou je znázorněn na obrázku Obrázek 3.3 - Rozdíl v typech záloh databáze. Po vytvoření úplné zálohy jsou v časech t1, t2, t3 a t4 provedeny oba typy záloh. Zatímco každá rozdílová záloha obsahuje změny všech dat od úplné zálohy, zálohy transakčního protokolu na sebe navazují. Podle toho se také někdy zaměňuje označení ‚záloha transakčního protokolu', za zálohu přírůstkovou.

---

[5] Ukázky příkazů pro práci se zálohami jsou převzaty z (12 stránky 199-222).



Obrázek 3.3 - Rozdíl v typech záloh databáze

3.6.1.4 Záloha skupiny souborů

Na rozdíl od předchozích záloh, které ukládaly všechny data v databázi (s rozdílem v jejich množství), můžeme u záloh skupin souborů definovat, kterou část databáze chceme zálohovat. Toto řešení lze efektivně využít ve chvíli, kdy máme v databázi množství dat, které jsou postradatelné či snadno nahraditelné, a jen malou část těch, které je třeba zálohovat. Pomocí zálohy skupiny souborů lze na zálohách ušetřit spoustu prostoru, který by zabíraly jiné typy záloh.

3.6.2 Obnovení databáze ze zálohy

Abychom mohli provést obnovu databáze do požadovaného stavu, je třeba nejprve provést obnovu z úplné zálohy. K té jsou totiž vázány ostatní zálohy, které nám mají pomoct přiblížit se požadovanému stavu. Z úplné zálohy lze obnovit i databázi, která již fyzicky neexistuje.

Před samotným obnovením databáze je doporučováno provést takzvané ‚zálohování konce protokolu', což není nic jiného, než vytvoření zálohy transakčního protokolu (9 str. 311). To lze provést samozřejmě jen ve chvíli, že protokol (soubor s příponou .ldf) nebyl poškozen či smazán. Účel tohoto kroku je zajištění co možná nejpřesnější obnovy databáze. Ve chvíli, kdy se podaří provést zálohování konce protokolu, máme k dispozici zálohu změn provedených od poslední zálohy stejného typu, případně od úplné zálohy.

Samotná obnova úplné zálohy se provede příkazem:

```
RESTORE DATABASE eLogika
FROM DISK = 'eLogika_1.bak'
WITH STANDBY = 'c:\tmp\eLogika_1.stn'
```

Za klíčovými slovy FROM DISK je uveden název souboru s úplnou zálohou databáze. V případě, že se soubor nenachází ve výchozím umístění, je třeba doplnit i cestu k souboru. Volitelně lze příkaz doplnit o možnost uchovat obraz obnovené databáze v pomocném souboru, ze kterého lze příkazem SELECT načítat data.

Obnovení rozdílové zálohy má naprosto shodnou syntaxi, jen je třeba vybrat soubor s rozdílovou zálohou (přípona .dif). Aby bylo možné databázi opět plně používat, je třeba dokončit proces obnovení příkazem:

```
RESTORE DATABASE eLogika
WITH RECOVERY
```

```
RESTORE LOG eLogika
FROM DISK = 'eLogika_1.trn'
WITH STANDBY = 'c:\tmp\eLogika_1.stn'
```

Obnovení záloh transakčního protokolu má syntaxi velmi podobnou.

Důležité je, v případě, že je záloh transakčního protokolu více, obnovovat je ve stejném pořadí, v jakém byly vytvořeny. Jako poslední zálohu lze obnovit ‚konec protokolu', jehož zálohu jsme vytvořili před začátkem obnovy databáze.

3.6.3 Návrh zálohování databáze pro systém eLogika

Aby bylo zajištěno, že v případě potřeby bude dostupná aktuální záloha databáze, nelze se spoléhat na ruční spouštění příkazů pro vytvoření zálohy. Celý proces je potřeba automatizovat, aby se minimalizovalo riziko nedostupnosti vhodné zálohy. Microsoft SQL Server disponuje službou SQL Server Agent, která je určena právě k automatickému provádění zadaných operací.

3.6.3.1 Použité typy záloh

Pro systém eLogika jsem zvolil kombinaci úplné zálohy (která je prakticky povinná) a zálohy transakčního protokolu. Úplná záloha bude prováděna jednou denně, vždy ve dvě hodiny v noci, kdy je nejpravděpodobnější, že systém nebude vytížen. Přírůstková záloha bude prováděna periodicky každé čtyři hodiny, počínaje čtvrtou hodinou ranní.

Názvy souborů záloh jsou generovány na základě data a času, aby bylo možné snadno nalézt nejaktuálnější úplnou zálohu a následně určit posloupnost přírůstkových záloh.

```
DECLARE @date varchar(20)
DECLARE @name varchar(30)
SELECT @date = CONVERT(varchar,DATEPART(YYYY,GETDATE()))+
               CONVERT(varchar,DATEPART(MM,GETDATE()))+
               CONVERT(varchar,DATEPART(DD,GETDATE()))+
               '-'+CONVERT(varchar,DATEPART(HH,GETDATE()))
SET @name = 'c:\elogika\backup\eLogika_' + @date + '.bak'
```

Vzhledem k tomu, že každá záloha bude mít unikátní název, nebudou se přepisovat ale hromadit a zabírat čím dál více místa na disku. Vzhledem k aktivované kompresi zálohových souborů a úložným možnostem dnešních počítačů, by to ovšem neměl být žádný problém. Pokud by se dané řešení projevilo jako neúnosné a nebylo by potřeba uchovávat zálohy dlouhodobě, dalo by se generování názvů upravit tak, aby se zálohy každý týden přepisovaly.

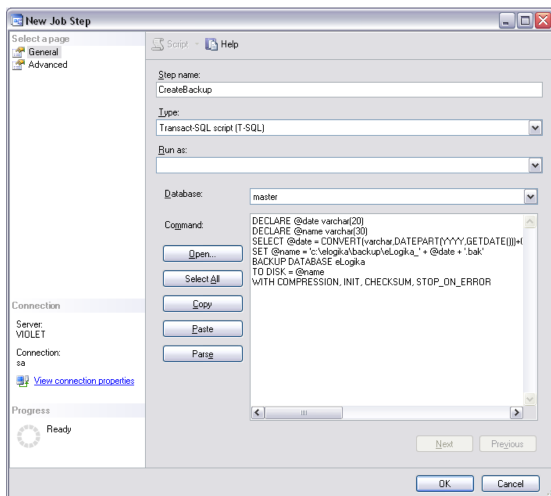
3.6.3.2 Nastavení zálohování

Postup, jak vytvořit nové úkoly pro službu SQL Server Agent popíšu z pohledu práce s SQL Server Management Studio. Vytvoření a spuštění nového úkolu lze provést i pomocí příkazů T-SQL, ovšem v tomto případě jde jen o spuštění několika systémových procedur.

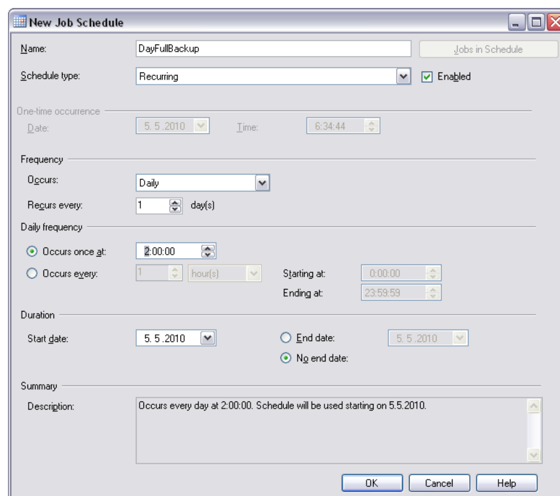
Prvním krokem je vytvoření nového úkolu, což provedeme z kontextové nabídky služby SQL Server Agent. Ta má svého zástupce na panelu Object Explorer. Na kartě *General* stačí vyplnit název úkolu, což může být třeba ‚eLogikaFullDbBackup'. Zajímavější je pak karta *Steps*, kde se nastavují jednotlivé kroky, které jsou v rámci daného úkolu provedeny. V dialogovém okně pro

vložení nového kroku je třeba vyplnit název kroku a zvolit typ kroku. V našem případě jde o SQL dotaz, takže typ je T-SQL. Do pole *Command* stačí vložit příkaz pro vytvoření zálohy a potvrdit (Obrázek 3.4 - Vytvoření kroku pro úkol SQL Server Agenta). Jako další krok je naplánování, kdy bude daný úkol spuštěn. To lze nastavit na kartě *Shedules*. V dialogovém okně pro vytvoření nového plánu je opět potřeba vyplnit název. Z dalších položek to je frekvence spouštění, což je denně a to v čase 2:00:00 (Obrázek 3.5 - Vytvoření časového plánu pro úkol SQL Server Agenta). Po uložení časového plánu stačí potvrdit celý úkol a ten nyní zajistí automatické vytvoření zálohy databáze každý den ve 2h.

Vytvoření úkolu pro přírůstkové zálohy se provede prakticky totožně.



Obrázek 3.4 - Vytvoření kroku pro úkol SQL Server Agenta



Obrázek 3.5 - Vytvoření časového plánu pro úkol SQL Server Agenta

3.7 Transparentní šifrování dat

Možnost, jak zabezpečit databázové soubory fyzicky uložené na disku, přináší nejvyšší verze Microsoft SLQ Server 2008. Tou možností je využití transparentního šifrování dat pro zvolenou databázi. Transparentní zde znamená, že šifrování i dešifrování probíhá v režii serveru a celý provoz zůstává nezměněn. Není třeba zadávat speciální klíč pro přístup k datům, ani upravovat aplikaci, aby zajistila dešifrování. Při požadavku na získání dat, jsou automaticky na serveru dešifrována a zaslána aplikaci.

Pro povolení transparentního šifrování je třeba udělat pár kroků. Prvním z nich je vytvoření hlavního klíče, tedy MASTER KEY v databázi master, a následné vytvoření certifikátu, kterým posléze zabezpečíme klíč k databázi eLogika.

```
USE [master];
GO
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'velmiTajneHeslo';
GO
CREATE CERTIFICATE eLogikaCertifikat
WITH SUBJECT = 'Certifikát k zašifrování databáze
```

Je nanejvýš důležité, vytvořený certifikát zálohovat a uložit bezpečně mimo server. Pokud by bylo potřeba databázi obnovit ze zálohy a certifikát by nebyl dostupný, došlo by ke ztrátě všech dat, jelikož by se nepodařilo databázi dešifrovat a připojit. Zálohu certifikátu společně s klíčem

provedeme příkazem:

```
BACKUP CERTIFICATE eLogikaCertifikat
TO FILE = 'C:\eLogikaCertifikatBackup.cer'
WITH PRIVATE KEY (FILE = 'c:\ eLogikaCertifikatBackup.key',
      ENCRYPTION BY PASSWORD = 'druheTajneHeslo');
GO
```

Nyní můžeme přejít do databáze eLogika, kde vytvoříme šifrovací klíč pro danou databázi. Zmíněný klíč je zabezpečený pomocí dříve vytvořeného certifikátu. Na závěr stačí upravit nastavení databáze tak, aby bylo transparentní šifrování povoleno.

```
USE [eLogika];
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_128
ENCRYPTION BY SERVER CERTIFICATE eLogikaCertifikat;
GO
ALTER DATABASE [eLogika]
SET ENCRYPTION ON;
GO
```

Pokud by nyní někdo odcizil databázové soubory a chtěl si databázi připojit ke svému serveru, tak se mu to bez certifikátu, který jsme dříve vytvořili, nepovede. Stejně tak pro obnovu databáze ze zálohy je potřeba zmíněný certifikát.

Transparentní šifrování má samozřejmě i své nevýhody. Fakt, že data musí být pro každý požadavek na databázi dešifrována, přináší jistou režii. Vzhledem k tomu, že v rámci bezpečnosti je při povolení transparentního šifrování některé databáze, zašifrována i databáze tempdb, může se zpomalení zpracovávání dotknout i jiných databází.

4 SQL Audit

Opatření, které jsem popsal v předchozích kapitolách, by se daly shrnout jako preventivní. Tedy takové, které mají zajistit vyšší bezpečnost dat uložených v databázi a řídit přístup k nim. Současně mají snížit možnost neoprávněného přístupu k SQL serveru. V této kapitole se budu věnovat metodě, kterou jsem navrhl kvůli možnosti odhalení, zda došlo k neoprávněnému přístupu k datům, případně, zda byl systém eLogika používán nestandardním způsobem.

Základ celé metody tvoří záznamy o operacích s daty, které požaduje systém eLogika po databázovém serveru. Podstatné pak jsou informace o tom kdo, kdy, s jakými daty a jakou operaci prováděl. Z těchto záznamů lze sestavit posloupnost operací, které jednotliví uživatele aplikace prováděli. Tuto posloupnost je pak možné porovnat se známými vzory posloupnosti operací a stanovit, zda jsou požadavky na databázový server zasílány tak, jak se očekává. Pokud by požadavky nepřicházely v posloupnosti, která odpovídá některému známému vzoru, bylo by zřejmé, že je neposílá systém eLogika, ale jde o možné napadení databáze.

Možností, jak tyto záznamy získat, je několik. Nejprostší, ovšem také nejnáročnější způsob vyžaduje spolupráci s aplikací, jejíž napadení chceme odhalit. Aplikace se sama musí postarat o hromadění záznamů o svém provozu. To vyžaduje úpravu volaných procedur, kde se doplní část pro logování prováděných akcí. V horším případě se musí upravit přímo kód aplikace tak, aby bylo logování zajištěno. To vše vyžaduje spoustu času na samotné zajištění potřebných záznamů, ale také značně ovlivní výkon aplikace, která musí zajišťovat operace nad rámec zamýšlené funkčnosti. Výhodou takového řešení je přenositelnost. Způsob shromažďování potřebných dat není závislý na žádné speciální technologii databázového systému a tak jej lze bez větších problémů použít při nasazení na jinou verzi či zcela jiný databázový systém.

Druhou možností, již méně náročnou jak na úpravu celé aplikace, tak na samotný výkon aplikace, je použití triggerů. Trigger je programová jednotka databázového systému, která je automaticky spouštěna při definované operaci nad určitou tabulkou. Shromažďování dat by tedy probíhalo prostřednictvím ukládání dat o spouštěných operacích nad danými tabulkami. To by bylo prováděno v tělech triggerů, které by byly vytvořeny pro všechny tabulky a možné operace nad nimi. Výhodou tohoto řešení je fakt, že o obsluhu triggerů se stará samotný databázový systém a jejich výkon je tedy optimalizován. Díky zavedení podpory triggerů do všech větších databázových systémů, je rovněž stále zajištěna přenositelnost tohoto řešení. Nevýhodou je až příliš velká detailnost získaných dat, která ovlivní složitost následné analýzy dat.

Poslední možností je využití technologií, které nabízí jednotlivé databázové systémy. Ty jsou optimalizovány tak, aby jejich použití co nejméně ovlivňovalo práci s databázemi. Většinou jsou také široce nastavitelné a množinu shromažďovaných dat tak lze velmi přesně specifikovat. Krom specifických omezení jednotlivých databázových systémů, má toto řešení jedinou vadu, kterou je nemožnost použití navrženého řešení na jiný databázový systém. Většina rozsáhlých projektů ovšem počítá s využitím konkrétních technologií a s nasazením na konkrétní platformu, pro kterou jsou vyvíjeny. Proto lze i systém pro shromažďování záznamu o operacích s daty, navrhnout pro konkrétní databázový systém, bez nutnosti uvažovat přenositelnost na jiný systém. Jednoduchá implementace a nejmenší možné zatížení databázového serveru, jsou důvody, pro které jsem si vybral právě tuto možnost.

4.1 SQL Audit v MSSQL 2008

Microsoft SQL Server 2008 ve své nejvyšší verzi, tedy Enterprise, umožňuje tvorbu uživatelem definovaného SQL auditu (10). Ten přináší možnost detailně sledovat operace prováděné jak nad celým serverem, tak i nad jednotlivými databázemi. Velkou výhodou je možnost sledovat pouze konkrétní akce nad konkrétními objekty a to i pro konkrétní iniciátory akce. Lze tedy sledovat například jen vkládání do jedné tabulky prováděné jedním uživatelem, ale i mazání ze všech tabulek konkrétní databáze, všemi uživateli, kteří jsou členy konkrétní role. Možnost velmi detailně nastavit co vše zaznamenávat, přináší velkou úsporu výkonu, ale i snižuje objem dat, které je nutné uchovávat.

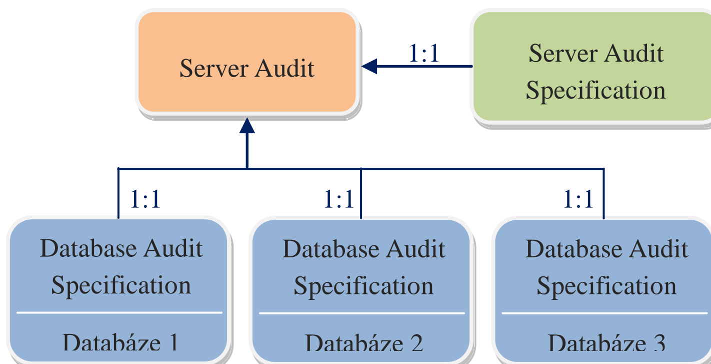
Celý SQL Server Audit byl navržen s ohledem na snadnou spravovatelnost. Všechny možnosti, které audit nabízí, lze nastavit a ovládat pomocí rozhraní SQL Server Management Studia (SSMS), programově pomocí Server Management Objects (SMO) nebo pomocí Transact-SQL (T-SQL). Poslední jmenovaný přístup umožňuje sestavit SQL dotazy, které vytvoří, nastaví a spustí celý audit. Ty lze následně použít na různých serverech, a proto jsem pro zprovoznění auditu zvolil právě tuto možnost.

4.1.1 Struktura auditu

Audit v SQL Server 2008 je rozdělen do tří hlavních objektů. Každý z nich pak popisuje nastavení některé části celého auditu.

- Server Audit – Definuje způsob a cíl zápisu auditních dat. Neurčuje však, jaké události mají být sledovány.
- Server Audit Specification – Definuje, které události v kontextu celého SQL serveru mají být auditem sledovány. Jde převážně o změny nastavení serveru, změny oprávnění na serverové objekty a správu uživatelů.
- Database Audit Specification – Definuje události v kontextu databáze, které mají být auditem sledovány. Jedná se tedy hlavně o události, přímo či nepřímo manipulující s daty uloženými v databázi. Druhou větší skupinu tvoří události upravující oprávnění pro přístup k datům.

Aby mohl audit fungovat, musí být vytvořen minimálně objekt Server Audit. Ten však sám o sobě nezpůsobí zápis žádných dat. K tomu je zapotřebí vytvořit minimálně jeden z objektů pro specifikaci událostí, na které se má audit zaměřit. Vztah mezi jednotlivými objekty je pak znázorněn na obrázku Obrázek 4.1 - Struktura auditu SQL Serveru.



Obrázek 4.1 - Struktura auditu SQL Serveru

Objekt Server Audit může mít přiřazen maximálně jeden objekt Server Audit Specification a libovolné množství objektů Database Audit Specification. Vazba mezi objekty je vždy 1:1. Na serveru pak může být vytvořeno libovolné množství objektů Server Audit.

4.1.2 Obsah auditních záznamu

Záznamy, které SQL audit produkuje, mají přesně danou strukturu. Ta je vytvořená tak, aby pokryla množství informací produkované všemi možnými událostmi, které lze do auditu zahrnout. Seznam atributů auditních záznamu je následující (11).

- event_time – datum a čas vyvolání sledované události
- sequence_no – v případě, že je informace o události příliš velká na to, aby se vlezla do jednoho záznamu, je rozdělena a označena identifikátorem pořadí
- action_id – identifikátor operace, která byla auditem sledována
- succeeded – příznak, zda byla operace úspěšná
- permission_bitmask – přidělená oprávnění na operaci (nemusí být vždy vyplněno)
- is_column_permission – příznak, zda byla oprávnění přidělena jen na konkrétní sloupce
- session_id – identifikátor relace, která událost vyvolala
- server_principal_id – identifikátor loginu přihlášeného k SQL serveru, který událost vyvolal
- database_principal_id – identifikátor uživatele databáze, který událost vyvolal
- object_id – identifikátor objektu, ke kterému se vztahuje operace sledovaná auditem
- class_type – třída, do které spadá objekt sledovaný auditem
- session_server_principal_name – název loginu přihlášeného k SQL serveru
- server_principal_name – aktuální login přihlášený k serveru
- server_principal_sid – identifikátor loginu přihlášeného k SQL serveru
- database_principal_name – název uživatele připojeného k databázi
- server_instance_name – název instance SQL serveru, na které k události došlo
- database_name – název databáze, v jejímž kontextu k události došlo
- schema_name – název schématu, v jehož kontextu k události došlo
- object_name – název objektu, ke kterému se událost vztahuje
- statement – SQL dotaz, který událost vyvolal
- additional_information – dodatečné informace o události, která je auditem sledována

Ne všechny atributy mají, u nejrůznějších operací sledovaných auditem, vyplněnu hodnotu. Rovněž ne všechny mají dostatečnou vypovídající hodnotu pro kontrolu auditu. Často se jedná o dvojice identifikátor – slovní popis, a tak se některé atributy hodí spíše pro automatické zpracování, jiné pro vizuální kontrolu výsledku auditu.

4.2 Nastavení auditu pro potřeby systému eLogika

Účel SQL auditu je shromáždění dat, ze kterých je následně možné určit, zda nedošlo k neoprávněné manipulaci s daty. Ta může nastat jak zneužitím systému eLogika, tak i přímým napadením databázového serveru. Proto je důležité, zaměřit se v auditu na obě tyto varianty a podrobit tak auditu všechny možnosti, jak k datům systému eLogika přistupovat.

Při nastavování auditu je ovšem důležité myslet i na množství dat, které bude shromažďováno. Příliš obecným nastavením auditu může dojít k zbytečnému zatěžování databázového serveru a také k hromadění dat, které nebudou v následné analýze vůbec použity.

Struktura příkazů pro vytvoření a nastavení auditu, byla převzata z (12 stránky 285-287).

4.2.1 Vytvoření auditu

Základním kamenem celého SQL auditu je objekt Server Audit. Ten určuje, kam mají být záznamy auditu zapisovány. Možnosti jsou Security nebo Application Log systému Windows, případně zápis do binárního souboru na disk. Kvůli jednoduchosti programového načtení záznamů a také udávané nejmenší zátěži systému, jsem zvolil ukládání do souboru. Tato volba vyžaduje zadání několika parametrů. Jsou jimi:

- FILEPATH – Cesta k cílovému adresáři pro uložení auditních záznamů.
- MAXSIZE – Maximální velikost auditního logu v MB, GB nebo TB. Druhá možnost je volba UNLIMITED. Minimální zadaná hodnota může být 2 MB a maximální 2.147.483.647 TB. Pokud není tento parametr uveden, je použita možnost UNLIMITED.
- MAX_ROLLVER_FILES – Maximální počet souborů pro auditní záznamy. Maximální zadaná hodnota je stanovena na 2.147.483.647. Také zde je možné použít volbu UNLIMITED, která je rovněž použita, pokud není tento parametr uveden. Dosáhne-li vytvořený počet souborů maxima, je automaticky smazán soubor s nejstarším datem.
- RESERVE_DISK_SPACE – Nastavení, zda se má prostor na disku, potřebný pro auditní záznamy, alokovat předem. Volby jsou ON a OFF, přičemž výchozí hodnota při neuvedení tohoto parametru je OFF. Tento parametr lze uvést pouze v případě, že předchozí parametry nejsou nastaveny na UNLIMITED.

Pro všechny tři možnosti uložení auditních záznamů lze nastavit další dva parametry, ovlivňující chování auditu.

- QUEUE_DELAY – Doba v milisekundách, do které se mají auditní záznamy zapsat do logu. Pokud je nastavena hodnota 0, jsou záznamy zapisovány synchronně s událostmi, které zápis vyvolaly. Výchozí hodnota je 1000, tedy jedna sekunda. Maximum lze nastavit na 2.147.483.647. Čím větší je doba vyhrazená pro zápis záznamů, tím menší je dopad auditu na celkový výkon databázového serveru.
- ON_FAILURE – Reakce na situaci, kdy nelze zapsat auditní záznamy do logu. Volby jsou CONTINUE, tedy pokračovat dále v činnosti, nebo SHUTDOWN, čili vypnutí dané instance serveru.

Server Audit je třeba vytvořit v kontextu hlavní databáze master, a po jeho vytvoření je třeba jej aktivovat. Případně lze atribut STATE, nastavit na hodnotu ON rovnou při vytváření auditu.

```
USE [master]
GO
CREATE SERVER AUDIT [Audit-elogika]
TO FILE (
   FILEPATH = N'C:\elogika\audit\', MAXSIZE = 50 MB
   ,MAX_ROLLOVER_FILES = 6, RESERVE_DISK_SPACE = ON
)
WITH (
   QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE, STATE = ON
)
GO
```

Hodnoty jako velikost a počet souborů pro zápis auditních dat, je třeba nastavit na základě zkušeností z provozu aplikace. Až po nasazení v ostrém provozu je možné odhadnout množství generovaných dat a dobu, za jak dlouho budou vyčerpány přidělené kvóty. Podle toho lze pak upravit nastavení auditu.

4.2.2 Specifikace serverového auditu

Události, které lze sledovat prostřednictvím nastavení v objektu Server Audit Specification, se vztahují vždy na všechny objekty dané třídy, v rámci celého SQL serveru (13). To znamená, že se v záznamech auditu objeví i události provedené vůči jiné databázi, než naší. V případě serveru, na kterém má databázi více projektů, to může mít za následek zbytečné hromadění nepoužitelných dat. U některých záznamů nelze ani určit, nad jakou databází byla událost vyvolána. V těchto případech může nastat problém při analýze auditních dat, kdy se událost vyvolaná nad jinou databází může jevit jako možné napadení databáze eLogiky. Z těchto důvodů je dobré, množství sledovaných událostí na serverové úrovni, omezit na nutné minimum.

Mezi takové události jsem zařadil neúspěšné přihlášení k SQL serveru spadající do třídy události FAILED_LOGIN_GROUP. Tato událost je vyvolána zadáním špatného hesla, či připojením uživatele ke koncovému bodu [3.4], na který nemá oprávnění. O připojení k databázi se stará výhradně systém eLogika a nelze tedy tuto událost vyvolat při běžném používání. Vyvolání této události je jasnou známkou neoprávněného přístupu k databázovému serveru. Podobná situace nastává v případě změny hesla pro připojení uživatele k databázi. Jakákoliv změna u přihlašovacích údajů je podezřelá. Třetí skupinou, kterou jsem do auditu zahrnul, je SERVER_PERMISSION_CHANGE_GROUP. Ta reaguje na změnu oprávnění k objektům v kontextu serveru, tedy například změnu vlastníka databáze.

Vzhledem k tomu, že se třídy událostí nevztahují na žádný konkrétní objekt, ani původce události, je vytvoření objektu Server Audit Specification velmi jednoduché. Stačí určit jeho jméno, zvolit Server Audit, který se postará o uložení auditních záznamů a pomocí klíčového slova ADD určit třídy, které si přejeme sledovat.

```
USE [master]
GO
CREATE SERVER AUDIT SPECIFICATION [SAS-elogika]
FOR SERVER AUDIT [Audit-elogika]
      ADD (SERVER_PERMISSION_CHANGE_GROUP),
      ADD (FAILED_LOGIN_GROUP),
      ADD (LOGIN_CHANGE_PASSWORD_GROUP)
WITH (STATE = ON)
GO
```

Jak jsem zmínil výše, tak auditní data z kontextu serveru nemusí pocházet z událostí, které nás zajímají. Proto budou mít tyto údaje při následné analýze jen doplňkový charakter.

4.2.3 Specifikace databázového auditu

Objekt Database Audit Specification je, co se týče získávání dat, nejdůležitější částí celého auditu. Zde se nastavují události, nad konkrétní databází a vyvolané konkrétním uživatelem, které mají být zahrnuty do auditu (13). Právě z těchto záznamů budou sestaveny posloupnosti operací, které uživatele vykonali.

Struktura příkazu pro vytvoření databázové specifikace je velmi podobná příkazu pro serverovou specifikaci. Rozdíl je v přidávání jednotlivých událostí pomocí klíčového slova ADD. Pomocí něj lze přidat třídu událostí, nebo jen konkrétní událost. V případě přidání konkrétní události, je navíc třeba určit databázi, tabulku či jiný objekt, ke kterému se událost vztahuje. Což záleží na tom, jak detailně chceme události specifikovat. Rovněž je třeba určit, na jaké iniciátory události má být audit aplikován. Iniciátorem může být konkrétní uživatel, případně aplikační, nebo databázová role.

Právě v této chvíli se naplno projeví síla použití databázových rolí a vlastních přihlašovacích údajů k databázi, pro každého uživatele systému eLogika. K databázi systému eLogika lze přistupovat pouze ve chvíli, kdy je uživatel členem některé databázové role.[6] Pokud tedy nastavíme audit tak, aby sledoval události spuštěné databázovými rolemi, zajistíme, že budou sledovány pro všechny uživatele systému eLogika. Díky použití jedinečných údajů pro připojení k databázi pro každého uživatele, pak lze z auditních záznamů vyčíst, který uživatel danou akci vyvolal. Tento údaj je zaznamenám v atributu server_principal_name. Díky tomu, že lze oddělit události jednotlivých uživatelů, můžeme sestavit pro každého z nich posloupnost operací, které v eLogice provedl.

Z výše uvedeného vyplývá, že jako iniciátor budou ve specifikaci databázového auditu uvedeny jednotlivé databázové role. Jako událost sledovaná auditem bude uvedeno EXECUTE, což je příkaz pro zavolání databázové procedury. A jak jsem zmínil dříve, takřka všechny požadavky na databázi jsou uskutečňovány voláním procedur. Tím audit zaznamená téměř kompletní komunikaci aplikace s databází.

Problém by nastal ve chvíli, když bych se pokusil zaznamenat i komunikaci, kterou provádí komponenta TableAdapter o které jsem se zmínil v kapitole 3.3.2. Nebyl by problém nastavit audit na sledování příkazu SELECT pro konkrétní tabulky. Ovšem audit MS SQL Serveru sleduje vyvolané události rekurzivně i v jiných událostech. Znamenalo by to, že by audit zaznamenal volání všech procedur, ale také všechny příkazy SELECT, které jsou v tělech procedur. Tím by výrazně narostlo množství shromažďovaných dat s minimálním přínosem pro bezpečnost.

Tímto jsou pokryty možnosti přístupu k databázi prostřednictvím systému eLogika. Zbývá tak ještě zahrnout do auditu události super administrátora serveru a dalších možných uživatelů, kteří nějak získají přístup k databázi. Přístup super administrátora ošetříme sledováním všech operací, které manipulují s daty pro uživatele dbo, pomocí kterého super administrátor k databázi přistupuje. Uživatele, kteří aktuálně nemají přístup k databázi, ale nějakým způsobem jej získají, nelze auditem sledovat, jelikož nevíme, o jakého uživatele půjde. Lze však sledovat operace pro udělení

---

[6] Výjimku tvoří přístup v roli majitele databáze, kterým je uživatel dbo. Uživatel dbo je pak svázán s loginem sa [3.5.1].

členství v některé z rolí databáze a tak odhalit, že se někdo pokouší přistupovat k datům. Sledování změny členství v rolích, zajistíme přidáním třídy DATABASE_ROLE_MEMBER_CHANGE_GROUP do specifikace auditu.

Na závěr jsem do specifikace auditu zahrnul třídu AUDIT_CHANGE_GROUP, která obsahuje události manipulující s auditem jako takovým. Pokud se jej tedy někdo pokusí vypnout, či změnit sledované události, samotný audit tyto činnosti zaznamená.

Příkaz pro vytvoření objektu Database Audit Specification je nutné spustit v kontextu databáze, kde chceme objekt vytvořit. V našem případě tedy eLogika.

```
USE [eLogika]
GO
CREATE DATABASE AUDIT SPECIFICATION [DAS-elogika]
FOR SERVER AUDIT [Audit-elogika]
      ADD (DATABASE_ROLE_MEMBER_CHANGE_GROUP),
      ADD (AUDIT_CHANGE_GROUP),
      ADD (DELETE ON DATABASE::[eLogika] BY [dbo]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [dbo]),
      ADD (INSERT ON DATABASE::[eLogika] BY [dbo]),
      ADD (SELECT ON DATABASE::[eLogika] BY [dbo]),
      ADD (UPDATE ON DATABASE::[eLogika] BY [dbo]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [elogika_system_R]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [elogika_sysadmin_R]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [elogika_tajemnik_R]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [elogika_garant_R]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [elogika_tutor_R]),
      ADD (EXECUTE ON DATABASE::[eLogika] BY [elogika_student_R])
WITH (STATE = ON)
GO
```

4.3 Příprava dat pro zpracování výstupu auditu

Aby bylo možné porovnat posloupnosti operací, které uživatelé prováděli, se vzorovými událostmi, je třeba mít připraven jejich aktuální seznam. Vzhledem k tomu, že data potřebná pro analýzu auditních záznamů nejsou součástí datového modelu systému eLogika, využívám pro jejich uchování vlastní databázi. Současně tak dochází k oddělení uživatelů databáze eLogika od potencionálně citlivých dat, shromážděných v rámci SQL auditu.

4.3.1 Příprava auditních dat

Pracovat se záznamy auditu uloženými v souborech, je velmi zdlouhavé. Obzvláště pokud chceme pracovat s velkým množstvím záznamů. Z toho důvodu jsou auditní záznamy, před samotným začátkem analýzy, načteny ze souborů do databáze. Tabulka pro uchování záznamů má stejnou strukturu, jakou mají samotné auditní záznamy. Podle času posledního uloženého záznamu v databázi, jsou z auditních souborů načteny jen nové záznamy.

Načtení času posledního záznamu v databázi. V případě, že jde o první načtení záznamů ze souboru, je čas nastaven na 1.1.1970-00:00:00.

```
DECLARE @time DATETIME2(7)
SELECT TOP(1) @time = event_time
      FROM log_items ORDER BY event_time DESC
IF @time IS NULL SET @time = '1.1.1970'
```

Pomocí systémového pohledu *server_file_audits*, jsou do proměnných *folder* a *auditguid*, načteny údaje o požadovaném auditu. Z nich je pak sestaven vzor pro cestu k souborům s auditními záznamy.

```
DECLARE @filepattern VARCHAR(300)
DECLARE @folder VARCHAR(300)
DECLARE @auditguid VARCHAR(40)
SELECT @auditguid = audit_guid, @folder = log_file_path FROM
      sys.server_file_audits WHERE name = 'Audit-eLogika'
SELECT @filepattern = @folder + '*_' + @auditguid + '*'
```

Uložení do databáze neprobíhá po jednotlivých záznamech, ale je využita možnost uložit celou načtenou množinu výsledků. Načtení dat probíhá klasickým způsobem, pomocí příkazu SELECT, s tím rozdílem, že jako zdroj není použita tabulka, ale funkce *fn_get_audit_file*. Ta má jako parametr výše zmíněný vzor cesty a názvu auditních souborů. Výstupem funkce je tabulka s auditními záznamy. Ty jsou navíc díky systémovým pohledům *dm_audit_class_type_map* a *dm_audit_actions* rozšířeny o detailnější popis k některým atributům.

```
INSERT INTO log_items
(
      event_time, class_type, class_type_desc,action_id, action_name,
      succeeded, session_id,server_principal_name,
      database_principal_name, database_name, schema_name, object_id,
      object_name
)
SELECT f.event_time, f.class_type, c.class_type_desc, f.action_id,
      a.name AS action_name, f.succeeded, f.session_id,
      f.server_principal_name,f.database_principal_name,
      f.database_name, f.schema_name, f.object_id, f.object_name
   FROM fn_get_audit_file(@filepattern, NULL, NULL) AS f
   JOIN sys.dm_audit_class_type_map c ON f.class_type = c.class_type
   JOIN sys.dm_audit_actions a ON f.action_id = a.action_id
      AND c.securable_class_desc = a.class_desc
   WHERE f.event_time > @time
   ORDER BY event_time
```

4.3.2 Seznam procedur

Data použitá k analýze jsou téměř výhradně záznamy o volání procedur. Právě ty tvoří základní jednotku celé analýzy. Aby mohly být procedury použity při dalším zpracování, je třeba mít k dispozici jejich seznam. Ten vznikal průběžně při postupném sbírání dat potřebných pro analýzu, avšak je možné seznam všech procedur získat také ze systémového pohledu sys.procedures v databázi eLogika.

Každá procedura má unikátní název v rámci databáze a také object_id, tedy identifikátor objektu, který je unikátní v rámci celého SQL serveru. Aby bylo možné identifikovat proceduru i po přenosu databáze na jiný server, případně přenesení dat z testovací databáze na provozní, musí být každá procedura opatřena vlastním identifikátorem, který je neměnný. Název procedury je pro další zpracovávání nevhodný a object_id se při vytvoření procedury v nové databázi změní, tedy by neidentifikoval správnou proceduru. Posledním atributem v seznamu procedur je popis, co daná procedura vykonává. Ten může být uživateli zobrazen v rámci výstupu analýzy. Každý záznam v seznamu procedur má tedy atributy: id_procedura, nazev, popis.

```
SELECT li.row_id, li.event_time, li.succeeded,
   li.database_principal_name, li.object_id, li.object_name,
   p.id_procedura
FROM log_items li INNER JOIN procedura p
   ON (li.object_name = p.nazev)
ORDER BY row_id DESC
```

Pro účely analýzy jsem použil jen některé atributy z auditních dat, ke kterým je ale připojen atribut id_procedura z tabulky procedury. Tím je zajištěno propojení auditních dat s identifikátory procedur eLogiky, díky kterému lze záznamy snadno analyzovat.

4.3.3 Seznam událostí

Událost je každá odezva aplikace na uživatelský podnět. Může to být přechod na novou stránku pomocí odkazu či tlačítka, dynamická úprava formuláře, refresh stránky nebo i použití tlačítka zpět či vpřed. Přesněji jde o událost tehdy, když uživatelem vyvolaná odezva vynutí zavolání jedné či více procedur. V systému jsou totiž i takové odezvy aplikace, které s databází nepracují a nespustí tak žádnou proceduru. Takovéto odezvy nelze z auditních záznamů identifikovat a dále je zpracovávat.

Důvodem pro zavedení událostí je snadná identifikace procedur, které jsou spouštěny v nestandardních situacích. Pokud je některá z procedur zavolána jiným způsobem, než v rámci vykonávání události, je zřejmé, že se tak stalo jinak, než prostřednictvím systému eLogika. Takovéto jednání pak svědčí o napadení systému, či o pokusu získání dat ze systému neoprávněným způsobem.

Jak jsem výše naznačil, událost je dána posloupností procedur. Nelze jí však podle ní identifikovat. Stejná posloupnost procedur může být vyvolána na různých místech aplikace a jde tedy vždy o jinou událost. Proto má každá událost přiřazen jedinečný identifikátor a také jedinečný název, který přibližuje, proč byla událost vykonána. Podrobněji to pak určuje popis události. Název události je složen ze čtyř částí, které dohromady jednoznačně identifikují danou událost a zároveň poskytnou dost informací pro odhadnutí jejího účelu. Na začátku je označení role, následuje označení původce události, její výstižný název a pořadové číslo. Role jsou značeny:

- SA – systémový administrátor
- TA – tajemník
- GA – garant
- TU – tutor
- ST – student

Původce události pak může být:

- M – událost je spuštěna z menu
- O – událost je vyvolána kliknutím na odkaz
- R – událost je vyvolána obnovením stránky

Každá událost je vázaná k jedné z rolí, které jsou v systému použity. Vzhledem k odlišné množině práv se občas stane, že stejná událost (myšlena jako odezva aplikace) vyvolá v rámci různých rolí, různou posloupnost procedur. Ve výsledku tedy jde o dvě různé události a jsou odlišeny rolí, ve které jsou použity. Existují i událostí, které jsou pro všechny role stejné. Kvůli snadnější analýze

jsem však pro každou roli vytvořil v seznamu událostí nový záznam.

4.3.3.1 Zápis posloupnosti procedur

Celý bezproblémový průběh analýzy závisí na tom, jak dokonale jsou popsány posloupnosti procedur jednotlivých událostí. K jejich popisu jsem si nakonec vybral syntaxi regulárních výrazů. Stejná událost může, v závislosti na množství dat v databázi, vyvolat různé posloupnosti procedur. Různé ovšem jen na první pohled. Ve skutečnosti je daná posloupnost procedur řízena určitým vzorcem opakování a podmínek, ze kterých pak vzejde výsledná posloupnost. Právě vzorec, podle kterého se daná posloupnost generuje, jde zachytit pomocí regulárního výrazu. Událost, která může být za různých stavů databáze, tvořena různými posloupnostmi procedur, tak je popsána pouze jedním regulárním výrazem.

Jako příklad uvedu zobrazení správy kurzů v roli tajemník. Část možných posloupností procedur je tato:

1 36 33 13 12 33 34 35	Jeden semestr v roce a jeden kurz s garantem
1 36 33 33 13 12 33 34 35 35	Dva semestry v roce a dva kurzy s garantem
1 36 33 13 12 33	Žádný semestr v roce, tedy ani žádný kurz
1 36 33 13 12 33 34	Jeden semestr v roce a žádny kurz s garantem
31 1 36 33 13 12 33 34	Jeden semestr v roce, žádný kurz s garantem a navíc zavoláno jako první událost po přihlášení

Tabulka 4.1 - Možné posloupnosti při různém stavu systému

Procedury odpovídající uvedeným ID jsou tyto:

1 - role_by_id_uzivatel_get

12 - skolainfo_by_id_get

13 - rok_get

31 - action_get

33 - semestr_get

34 - kurz_get

35 - garant_by_id_kurz_get

36 - rok_skola_get

Všechny tyto možné posloupnosti a další, které zde nejsou uvedené, lze popsat jedním regulárním výrazem, který vypadá následovně: (31)?1 36(33)+ 13 12 33(34(35)*)+

4.3.3.2 Struktura regulárních výrazů

Krom identifikátorů procedur, mají všechny symboly ve výrazech, popisujících posloupnosti procedur, vlastní význam. Nutno podotknout, že výrazy nejsou sestaveny ve své nejminimalističtější podobě, ale v takové, která je přijatelná pro proces analýzy auditních dat.

Kulaté závorky ohraničují posloupnost procedur, na kterou se vztahuje operátor, umístěný

bezprostředně za uzavírající závorkou. Daným operátorem může být:

- ? [otazník] – Výraz v závorce je v řetězci přítomen jednou nebo vůbec.
- + [plus] – Výraz v závorce je v řetězci přítomen jednou nebo vícekrát, a to bezprostředně za sebou.
- * [hvězdička] – Výraz v závorce není v řetězci přítomen vůbec, nebo libovolně-krát. Opět bezprostředně za sebou.

Posloupnosti ohraničené závorkami s operátory, do sebe mohou být libovolně zanořené.

Obsah kulaté závorky může být oddělen symbolem ‚|' tedy svislou čarou. Ta v regulárních výrazech značí alternativu. Výraz (A|B|C) tedy popisuje tři možné řetězce, přičemž každý obsahuje jen jeden výskyt jednoho z písmen A, B nebo C. Závorku může (ale nemusí) následovat jeden z výše zmíněných operátorů.

Pro účely popisu některých posloupností, jsou využity takzvané pojmenované skupiny. Jejich účel spočívá v tom, že si určíme nějakou posloupnost, kterou pojmenujeme, a následně ji můžeme celou použít v další části výrazu. Hlavní přínos je v možnosti opakovaně pracovat s naprosto stejnou, předem nepřesně definovanou posloupností.

Struktura pojmenované skupiny je následující: (?<nazev>posloupnost)(\k<nazev>). Ihned po otevírající kulaté závorce následuje otazník a v ostrých závorkách definovaný název. Ten pojmenovává posloupnost, která následuje za uzavírající ostrou závorkou a končí uzavírající kulatou závorkou. Posloupnost, kterou jsme si takto uložili, můžeme pak kdykoliv použít po zadání \k s názvem posloupnosti v ostrých závorkách. To vše uzavřeno do kulatých závorek.

Způsob a účel použití pojmenovaných skupin je patrný z následujícího příkladu.

Mějme písmena A, B a C, která tvoří posloupnost takovou, že začíná písmenem A, končí písmenem C a mezi nimi je libovolný počet písmen B (třeba i žádné). Takovou posloupnost popíšeme výrazem: A(B)*C. Dále chceme, aby za takovou posloupností následovalo písmeno D a za ním stejná posloupnost, jako před ním. Nabízí se možnost použít výraz A(B)*CD A(B)*C. Takový výraz popíše posloupnost ABBCDABBC, což chceme, ale popíše i ACDABBC, což už nechceme. Řešením je výraz: (?<w>A(B)*C)D(\k<w>). Po té, co víme, jak vypadá posloupnost v první závorce, uloží se pod jméno ‚w' a takto uloženou posloupnost použijeme v druhé závorce.

V několika málo posloupnostech je potřeba využít podmínku. Ta se zapisuje pomocí otazníku, který předchází kulaté závorce, ve které je podmínka rozhodující o dalším průběhu. V případě splnění podmínky, platí pro další běh obsah první kulaté závorky za podmínkou. Za svislou čárou, tedy alternativou, je pak v kulaté závorce uveden výraz, kterým se pokračuje v případě nesplnění podmínky. Celá podmíněná posloupnost by šla zapsat zhruba takto: (?(podminka)(splnena)|(nesplanena)). Abych mohl použít podmínku pro popis složitých posloupností, musel jsem ji skombinovat s pojmenovanou skupinou. Následující výraz kombinuje obě výše popsané struktury: ((?(w)((B)+)|(?<w>(A)+))C)+. Při prvním průchodu výrazem je proměnná pojmenované skupiny ‚w' prázdná, platí tedy obsah druhé kulaté závorky. Posloupnost rozpoznávaná tímto výrazem tedy musí začínat písmeny ‚A', následovanými jedním písmenem ‚C'. Při každém dalším průchodu je pak proměnná ‚w' naplněna sekvencí písmen ‚A' z prvního průchodu, a je tak platná první kulatá závorka, tedy posloupnost písmen ‚B' zakončena opět písmenem ‚C'. Rozpoznávaná posloupnost tak může vypadat například takto: AAACBBCBBBBC.

Stejného výsledku by v tomto případě šlo dosáhnout i jednodušším výrazem. U celkově složitějších výrazů si ovšem tímto způsobem lze výrazně zjednodušit výsledný zápis.

Poslední symbol použitý v regulárních výrazech je mezera. Ta se chová jako jakýkoliv jiný znak a tak je třeba zacházet s nimi opatrně. Z důvodu následné analýzy, musí výsledný výraz popisovat posloupnost identifikátorů procedur, oddělenou mezerami a zakončenou jednou mezerou.

4.3.3.3 Řazení událostí

Analýzu záznamů jsem založil na nalezení událostí v posloupnosti procedur. Jejich nalezení je umožněno díky regulárním výrazům, které popisují jednotlivé události. Pokud regulární výraz odpovídá posloupnosti procedur[7], je daná část posloupnosti procedur, kterou popisuje regulární výraz, identifikována jako konkrétní událost. Vyhledávání další události pak probíhá v posloupnosti, ze které byla odebrána část odpovídající předchozí nalezené události.

Často se stane, že dvě (nebo více) událostí mají shodnou počáteční část posloupnosti procedur, a liší se třeba jen v jedné proceduře na konci posloupnosti. Ještě častěji se pak stane, že posloupnost procedur jedné události odpovídá části posloupnosti události jiné. V takovém případě může lehce dojít ke špatnému určení události, která vyvolala danou posloupnost procedur. Ještě horší situace nastane ve chvíli, kdy posloupnost procedur zkrácená o předchozí nalezenou událost, neodpovídá žádné další události. Kvůli špatně určené události tak vznikne neidentifikovatelná část posloupnosti procedur, která ve výsledku může znamenat možné napadení databáze.

Příkladem pro takovouto situaci může být posloupnost identifikátorů procedur: *1 1 36 33 33 13 12 33 34 35*, které odpovídá událost pro zobrazení formuláře pro přidání garanta, s regulárním výrazem: *1 1 36(33)+ 13 12*. Po nalezení této události zůstane z původní posloupnosti jen *33 34 35*, což neodpovídá žádné události. Původní posloupnosti procedur však odpovídá i událost výpis kurzů pro rozdělení roku s regulárním výrazem: *1 1 36(33)+ 13 12 33 34 (35)**. Tato událost odpovídá celé posloupnosti procedur.

Z příkladu je jasné, že je výhodnější, při nalézání správné události, upřednostnit ty, které obsáhnou co největší část posloupnosti procedur, ve které události vyhledáváme. To ovšem není jednoduché. Regulární výrazy procedur mohou v různých případech odpovídat různě dlouhým posloupnostem. Proto jsem pro jejich upřednostňování použil minimální délku posloupnosti, kterou je schopen regulární výraz nalézt. Čím větší minimální délka, tím větší úsek v posloupnosti procedur může událost pojmout. Minimální délku stanovují ty procedury, které musí být vždy součástí posloupnosti, aby byla identifikována jako konkrétní událost. Jde tedy o ty procedury, které nejsou uzavřeny v kulatých závorkách, případně je operátor za závorkou +. Čili jeden nebo více výskytů obsahu závorky. Minimální délka pro první událost v předchozím příkladu je tedy 6, délka pro druhou událost je 8.

Aby bylo možné zohlednit i procedury, které se v události nemusí vyskytovat vždy, je jako druhé kritérium pro seřazení události použita délka regulárního výrazu. Pokud mají dvě události stejnou minimální délku, je pravděpodobnější, že větší část posloupnosti procedur obsáhne událost s delším regulárním výrazem.

---

[7] Pro porovnávání pomocí regulárních výrazů je posloupnost procedur reprezentována zřetězením identifikátorů procedur, oddělených mezerami.

Seznam událostí, setříděných sestupně podle minimální délky a počtu znaků regulárního výrazu, vrací procedura *udalost_get_all*, jejíž tělo tvoří výběrový dotaz:

```
SELECT * FROM udalost WHERE pouzit = 1
   ORDER BY [minimalni_delka] DESC, LEN([posloupnost_proc]) DESC
```

4.3.3.4 Události nulté úrovně

Některé události jsou označeny příznakem ‚nultá úroveň'. To značí, že jsou v eLogice, při přihlášení pod konkrétní rolí, dostupné v každém okamžiku práce s aplikací. Jde tedy o události, které lze spustit převážně z menu. Například pro roli systémový administrátor jsou události nulté úrovně tyto: zobrazení úvodní stránky, zobrazení vlastního profilu, správa licencí, správa škol, správa tajemníků a správa uživatelů.

Smysl tohoto příznaku je ve zmenšení objemu dat, které je třeba současně zpracovávat, při vyhledávání aktivit. Účelem tedy je snížit výpočetní náročnost celé analýzy. Podrobněji se o účelu událostí nulté úrovně zmíním v následujících kapitolách.

4.3.4 Seznam Aktivit

Aktivita je posloupnost událostí. Jde tedy o sled akcí, které vedou k uskutečnění zamýšlené operace. Tou může být výpis studentů z konkrétní třídy, vytvoření kurzu, nebo odstranění školy ze systému.

Přínos aktivit je v možnosti určit, zda se uživatel v rámci aplikace ‚pohybuje' standardním způsobem. Tedy zda mezi jednotlivými stránkami (a operacemi, které stránka nabízí) přechází pomocí ‚cest', které mu aplikace nabízí. Cestou je myšleno použití odkazu (tlačítka), přesměrování vynucené aplikací, nebo funkce webového prohlížeče pro pohyb v historii stránek. Samotné nevyužití některé z cest, nemusí být pokus o napadení aplikace, ale rozhodně jde o nestandardní chování a je dobré o něm informovat.

Seskupení událostí do aktivit umožní rozpoznat ruční zadání URL do prohlížeče a tedy přechod do části aplikace, která není z dané stránky dostupná. Druhou situací, kterou lze rozpoznat, je práce s aplikací ve více oknech prohlížeče.

Stejně jako události, má i každá aktivita přiřazen jednoznačný identifikátor a název. Struktura názvu je podobná té z události. Skládá se tedy z identifikace role, pod kterou spadá daná aktivita, stručného popisu aktivity a pořadového čísla. Dalšími atributy jsou role, popis aktivity a hlavně pak posloupnost událostí s jejich délkou.

4.3.4.1 Zápis posloupnosti událostí

Struktura zápisu posloupnosti událostí je shodná se zápisem posloupnosti procedur. Opět je použit zápis pomocí regulárních výrazů, který je popsán v kapitole 4.3.3.2.

Posloupnosti událostí jsou specifické tím, že ve většině případů existují dvě události, kterými může celá aktivita začínat. Jednou z nich je událost nulté úrovně [4.3.3.4] a druhou je událost, která je dostupná po vykonání některé operace, kterou nabízí stránka. Jde především o sdělení o (ne)úspěchu provedení operace, obsahující odkaz pro přechod na jinou stránku. Právě zobrazení této stránky může být první událostí některé z aktivit.

Aktivita musí brát v potaz také to, že každá událost, která je v aktivitě obsažena, může být na základě požadavku uživatele vykonána opakovaně. Opakované vykonání události je umožněno obnovením stránky prostřednictvím webového prohlížeče, které způsobí opakování i těch událostí, které jsou prováděny jen v rámci jedné stránky. V rámci aktivit rozlišuji tři druhy opakování událostí:

1. Opakování události vyvolá stejnou posloupnost procedur, jaká popisuje původní (opakovanou) událost.
   To se projeví jako spuštění stejné události několikrát po sobě. V rámci zápisu posloupnosti je opakování takovéto události zapsáno pomocí operátoru pro jeden a více výskytů. Opakovaná událost s identifikátorem 21 je tedy zapsána pomocí výrazu (21)+.
2. Opakování události vyvolá jinou posloupnost procedur, než původní opakovaná událost.
   V posloupnosti procedur se tato možnost projeví jako dvě různé události. Pro každou událost, která má při opakování odlišnou posloupnost procedur, je vytvořena nová událost se stejným názvem, ve kterém je jen změněn příznak původce události na ‚R'. Příklad zápisu takové události by mohl být výraz 22 (23)*. Operátor hvězdička, tedy žádné nebo více výskytů, je zde použit proto, že k opakování události může dojít jen někdy.
3. Opakování události vyvolá proceduru *role_by_id_uzivatel_get*, která má identifikátor 1.
   Tato situace nastává většinou při obnovení některého vstupního formuláře, k jehož zobrazení nejsou potřeba žádná data z databáze. Pro každou roli je vytvořena jedna událost popisující tuto situaci. Pro roli systémového administrátora je to událost s id 289 a opakování události by tedy mohlo být popsáno výrazem 24 (289)*.

Všechny události v aktivitách jsou tedy pomocí výše zmíněných zápisů, zahrnuty včetně možnosti opakování každé z aktivit.

Kvůli lepším možnostem nalezení aktivit v posloupnosti událostí vzniklé analýzou posloupnosti procedur, jsem pro každou aktivitu použil dva různé zápisy posloupnosti událostí (dva regulární výrazy). První z nich je takzvaný ‚originál', což je zápis odpovídající dosud zmíněným podmínkám. Druhý zápis je ‚upravený', přičemž úprava spočívá v zahrnutí možnosti, pohybovat se v rámci aktivity zpět pomocí historie stránek webového prohlížeče, případně aktivitu nedokončit (například aktivita vložení otázky bez jejího finálního uložení).

Tento druhý zápis je rozšířením zápisu originálního. Obsahuje tedy všechny identifikátory událostí jako originál, ale přidává navíc operátory, které zajistí akceptaci zpětného pohybu v posloupnosti událostí a možné nedokončení aktivity. Postup, jak upravit originální zápis, lze zobecnit do následujících kroků:

1. První část výrazu, která je společná pro více aktivit, zůstane beze změny a v dalším zpracovávání se s ní nepočítá. Taková část je například výběr konkrétní položky z menu. Tu obsahuje více aktivit a většinou jde také o samostatnou aktivitu.
2. Z levé strany je výraz uzavírán do závorek, ke kterým je přiřazen operátor pro jeden nebo žádný výskyt (otazník). Vždy je uzávorkována jen taková část, aby byla zahrnuta jedna událost, která buď operátor přiřazen nemá, nebo má operátor pro minimálně jeden výskyt (plus). Uzávorkování probíhá až do doby, kdy není uzávorkována jen jedna událost s minimálně jedním výskytem (většinou první událost zprava). V tomto okamžiku lze, díky operátoru pro žádný výskyt, přerušit aktivitu po jakékoliv události

3. Při pohybu zpět v historii stránek webového prohlížeče, nedochází k novému požadavku o zobrazení stránky. Ta je zobrazována díky cache paměti prohlížeče. V konečném důsledku nedochází ke spouštění žádných událostí. V rámci aktivity si to tedy lze představit tak, že se v posloupnosti událostí lze kdykoliv vrátit o pár kroků zpět a pokračovat následujícími událostmi, které již jednou byly provedeny. Je tak potřeba do výrazů popisující aktivity, zanést cykly. To se provede tak, že se celý výraz vloží do závorky s operátorem jednoho či více výskytů (plus). Pokud výraz obsahuje alternativy, je rovněž každá z nich uzávorkována a opatřena operátorem plus. Díky tomu, že lze celou posloupnost opakovat dle libosti a současně je možné dané opakování kdykoliv ukončit, je zajištěn libovolně dlouhý cyklus a tedy i pohyb zpět v historii prohlížeče.

Jako příklad uvedu úpravu výrazu popisujícího editaci akademického roku v roli tajemník. Originální výraz má podobu: ((56)+|87)88 (296)*89 (296)*. Význam jednotlivých událostí je:

- 56 – Výpis akademických roků školy (z menu)
- 87 – Výpis akademických roků školy (z výpisu o ne/úspěchu operace)
- 88 – Zobrazení formuláře pro editaci akademického roku
- 89 – Uložení změněných údajů o akademickém roku
- 296 – Obnovení stránky v roli tajemník

Dle kroku jedna zůstane začátek výrazu nezměněn. Výpis akademických kroků lze provést pomocí dvou různých událostí a ty se jako alternativy vyskytují ve více aktivitách. Druhý krok říká, že se mají uzávorkovat události, které se v posloupnosti vyskytují vždy. Výraz bude tedy vypadat: ((56)+|87)88 (296)*(89 (296)*)?. Událost 88 je poslední neuzávorkována a tak přecházíme k třetímu kroku. Celou upravovanou část opět uzávorkujeme: ((56)+|87)(88 (296)*(89 (296)*)?)+.

Takto upravený výraz popisuje jak normální průběh editace akademického roku, tedy posloupnost událostí 56 88 89, tak i obnovení stránky po některých událostech: 56 56 88 296 296 89. Stejně tak popisuje přerušenou aktivitu třeba před uložením upravených údajů: 56 88 a nakonec i aktivitu, v jejímž průběhu se uživatel vracel v historii prohlížeče: 56 88 296 89 88 89. Konkrétně tato posloupnost popisuje zobrazení seznamu akademických roků, výběr jednoho z nich pro editaci, obnovení stránky, uložení upravených údajů, návrat až na seznam akademických roků a opět výběr jednoho roku pro editaci a uložení údajů.

Výše uvedený postup ovšem není dokonalý. V upravené posloupnosti mohou vznikat zbytečná uzávorkování, které lze třeba odstranit spojením operátorů ? a + do operátoru *. Rovněž tento postup nelze aplikovat na některé velmi specifické aktivity.

Pokud by u aktivit nebyly použity dva různé zápisy posloupnosti událostí, ale pouze upravený zápis, mohlo by docházet k jejich předčasnému a tedy potencionálně chybnému ‚nalezení‘. Spousta aktivit je definována shodnými počátky posloupností událostí a v upraveném zápisu je většina z celé posloupnosti nepovinná. Proto by byla na analyzovaném úseku událostí prohlášena za odpovídající aktivitu (z dané množiny aktivit se shodným počátkem posloupnosti událostí) ta, která by byla kontrolována nejdříve. Záleželo by tedy na jejich seřazení. To by ovšem nebylo možné, vzhledem k velké podobnosti aktivit, dostatečně přesně stanovit.

4.3.4.2 Řazení aktivit

Pro řazení aktivit platí to samé, co pro řazení událostí. Pro následnou analýzu je výhodnější, pokud jsou upřednostněny aktivity obsahující co nejvíce událostí. Informace o minimálním počtu události v aktivitě, je opět v atributu délka. Vzhledem k tomu, že aktivita obsahuje dvě odlišné posloupnosti, jsou i atributy pro délku dva. Aby bylo možné seřadit všechny aktivity podle minimální délky jejich posloupností, a to nezávisle na tom, zda jde o posloupnost originální, nebo upravenou, je potřeba zahrnout každou aktivitu do výsledné množiny dvakrát, pokaždé s jinou posloupností.

Dosáhnout takovéhoto výstupu z jedné tabulky, je možné pomocí příkazu UNION, který kombinuje výsledky dvou či více dotazů. Jedním z těch dotazů je výběr originální posloupnosti a jí odpovídající minimální délka, druhým je pak upravená posloupnost s její minimální délkou. Kvůli odlišení originálních a upravených posloupností, je do výsledného výstupu přidán atribut originál. Výsledná množina záznamů je následně seřazená sestupně podle minimální délky a délky regulárního výrazu. V případě shodné minimální délky i délky výrazu u originální i upravené posloupnosti, je zvýhodněna posloupnost originální.

```
SELECT * FROM (
   SELECT aktivita_id, nazev, popis, posloupnost_udalosti_original
          AS posloupnost_udalosti, delka_original AS delka, [role],
          pouzit, 1 AS original
      FROM aktivita WHERE pouzit = 1
   UNION
   SELECT aktivita_id, nazev, popis, posloupnost_udalosti_upravena
          AS posloupnost_udalosti, delka_upravena AS delka, [role],
          pouzit, 0 AS original
      FROM aktivita WHERE pouzit = 1
) a
ORDER BY delka DESC, LEN(posloupnost_udalosti) DESC, original DESC
```

4.4 Analýza auditních záznamů

Celá analýza probíhá ve dvou hlavních krocích. V tom prvním je z posloupnosti procedur, která vychází z auditních záznamů, sestavena posloupnost událostí. V té je pak v druhém kroku vyhledána posloupnost aktivit, která je finálním výstupem celé analýzy.

4.4.1 Technologie LINQ

Před popisem samotné analýzy bych se chtěl zmínit o technologii LINQ (Language INtegrated Query), kterou v jejím průběhu hojně využívám. LINQ přináší zcela nové možnosti pro práci s daty v prostředí .NET Framework. LINQ je implementován pro práci s různými zdroji dat. Já používám implementaci LINQ to Objects, která je určena pro práci s kolekcemi uchovávanými v paměti. Syntaxe LINQ je velmi podobná dotazovacímu jazyku SQL a snadno tak lze provádět nejrůznější operace s objekty uloženými v některé z generických kolekcí. Více o technologii LINQ můžete dovědět na wikipedii (14).

Možnosti LINQu využívám při přetřiďování, případně omezování množiny záznamů v seznamech událostí a aktivit. Stačí tyto seznamy načíst z databáze na začátku analýzy, a dále je modifikovat pouze v paměti pomocí LINQu. Vzhledem k tomu, že pro každý cyklus v rámci analýzy jsou

potřeba plnohodnotné seznamy, patřičně setříděné, probíhají modifikace seznamů na jejích kopiích. V každém novém cyklu analýzy jsou pak upravené seznamy obnoveny do původní podoby. To vše bez dalších požadavku na data z databáze.

4.4.2 Použité objekty

Abych zpřehlednil následující popis průběhu analýzy, uvedu třídy použité při analýze, v této kapitole. Obecně by se daly rozdělit do tří skupin.

První skupina tvoří vrstvu pro komunikaci s databází. Pro každou tabulku v databázi, jejíž obsah je potřeba pro analýzu, je vytvořena jedna třída, zajišťující manipulaci se záznamy dané tabulky. Vzhledem k tomu, že při spuštění analýzy jsou všechna data pro analýzu již připravena, jsou v těchto třídách podstatné pouze metody pro načtení záznamů z databáze. Výstupem těchto metod jsou kolekce objektů, kde každý objekt je mapován na jeden záznam z databázové tabulky. Tyto objekty jsou instance tříd spadajících do druhé skupiny. Ta by se dala obecně shrnout jako množina objektů, která uchovává data v průběhu analýzy.

Mapování obsahu jednotlivých tabulek na objekty, je uveden v tabulce **Chyba! Nenalezen zdroj odkazů.**. První sloupec určuje tabulku, jejíž obsah chceme načíst. Druhý k ní přiřazuje třídu, která zajišťuje manipulaci s daty dané tabulky, a třetí určuje třídu, na kterou jsou mapovány jednotlivé záznamy z tabulek.

Tabulka	Třída databázové vrstvy	Třída objektu pro analýzu
log_items	AuditLogDB	AuditLogAnalyseClass
procedura	ProceduraDB	ProceduraClass
udalost	UdalostDB	UdalostClass
aktivita	AktivitaDB	AktivitaClass

Tabulka 4.2 - Mapování záznamů z databáze na objekty

Do skupiny objektů, uchovávajících data v průběhu analýzy patří dále třídy UserAnalyseClass, DostupnaUdalost a VariantaPosloupnosti. První jmenovaná slouží k uchování dílčích posloupností procedur pro jednotlivé uživatele. DostupnaUdalost obaluje událost, která byla nalezena v posloupnosti procedur, a přidává k ní příznak, zda je daná událost platná, nebo neplatná. Smyslu takovýto ‚obal‘ nabývá ve chvíli, kdy je pro danou posloupnost procedur nalezeno více událostí[8], a je třeba zaznamenat, která z nich je aktuálně považována za platnou. Nutnost použití objektu VariantaPosloupnosti vyplývá z výše uvedené možnosti různých událostí. Ve chvíli, kdy posloupnost událostí obsahuje alternativy, vznikne také více možností pro sestavení posloupnosti aktivit. Právě různé verze posloupnosti aktivit jsou zastřešeny objektem VariantaPosloupnosti. Z těchto variant je pak vybrána jedna, která je prohlášena za správnou.

Poslední skupinou jsou objekty, které jsou navrženy pro použití v komponentě GridView, a slouží tedy k prezentaci výsledku analýzy. Tyto objekty jsou specifické tím, že každý atribut, který

[8] Tedy, že danou posloupnost procedur popisuje regulární výraz několika různých událostí.

chceme v GridView vypsat, musí mít vytvořenu Property a nastavenu viditelnost na public. Další podmínkou je, aby byl zobrazovaný atribut primitivního datového typu.

4.4.3 Sestavení posloupností procedur

Celá analýza začíná rozdělením dat, nashromážděných auditem, podle jednotlivých uživatelů. O to se stará metoda getUzivatele, vracející kolekci objektů UserAnalyseClass. Každá instance pak náleží jednomu uživateli, který má v auditních datech alespoň jeden záznam. Seznam uživatelů je vytvářen postupně při procházení záznamů shromážděných auditem.

Při načtení každého záznamu z auditu je zkontrolováno, zda je původce záznamu[9] již obsažen v kolekci se záznamy uživatelů. Pokud ne, je vytvořena nová instance UserAnalyseClass, která je přidána do dané kolekce. Po nalezení záznamu daného uživatele, případně jeho vytvoření, je záznam z auditu přidán do dílčí posloupnosti daného uživatele. Současně je aktualizována řetězcová podoba posloupnosti procedur, která je složená z identifikátorů procedur oddělených mezerami.

Takto jsou zpracovány a rozděleny všechny záznamy z auditu, a kolekce s dílčími posloupnostmi uživatelů je předána k dalšímu zpracování. To probíhá tak, že je pro každého uživatele spuštěna metoda zpracujLog, která z posloupnosti procedur sestaví posloupnost událostí a z té je v metodě vyhodnotAktivity sestavena posloupnost aktivit. Ta je finálním výstupem celé analýzy auditních záznamů.

4.4.4 Nalezení událostí

Účelem metody zpracujLog je nalézt, v posloupnosti procedur konkrétního uživatele, vzájemně navazující části, které odpovídají dříve specifikovaným událostem. Procházení celé posloupnosti končí ve chvíli, kdy je dosaženo jejího konce.

Z důvodu rychlejšího zpracovávání celé posloupnosti, je ta rozdělena na menší úseky, které se zpracovávají postupně. Vzhledem k tomu, že sled procedur nemá bez znalosti konkrétních příčin jejich vyvolání (ty právě chceme zjistit) žádnou známou strukturu, zůstává tak jediným faktorem, podle kterého lze posloupnost rozdělit, čas vyvolání procedury. Pokud vezmeme v potaz fakt, že běžná událost vyvolá spuštění všech procedur, které obsahuje, ve velmi krátkém časovém rozmezí, lze podle prodlevy mezi jednotlivými procedurami určit hranici mezi koncem jedné a začátkem druhé události. I v případě rychlého vyvolávání událostí uživatelem lze říct, že časová prodleva mezi procedurami v rámci jedné události bude menší, než prodleva mezi procedurami různých, po sobě jdoucích událostí. V ideálním případě lze celou posloupnost procedur rozdělit na části, z nichž každá bude odpovídat přesně jedné události. Problém ovšem nastane ve chvíli, kdy budou procedury v rámci jedné události volány s větší časovou prodlevou, než je námi definovaná. Tato situace může nastat při přetížení serveru, případně ve chvíli, kdy vykonání některé procedury trvá neobvykle dlouho. Pokud tato situace nastane, bude posloupnost procedur rozdělena na menší úseky na nevhodných místech a při jejich následné analýze dojde k selhání, což povede k mylně identifikovanému napadení databáze. Nastavení větší časové prodlevy způsobí větší zátěž při analýze, ale výrazně sníží možnost chybné identifikace napadení. Určení optimální velikosti

[9] Uživatel, který provedl v systému eLogika operaci, která způsobila daný záznam v auditu.

prodlevy je možné až na základě výsledku zátěžových testů.

Seznam událostí, které se snažím v posloupnosti procedur identifikovat, je přímo z databáze dodán setříděný podle velikosti a složitosti regulárních výrazů, které události popisují. Díky tomu je proces nalezení správné události značně zefektivněn. Dalšího zlepšení lze dosáhnout ve chvíli, kdy budou při vyhledávání události upřednostněny ty, které mají v závislosti na předchozích událostech větší pravděpodobnost, že budou danou posloupnost procedur popisovat. Údaj, který lze pro takovéto upřednostnění použít je role, pro kterou je událost definovaná. Pokud se uživatel přihlásí do systému eLogika, provede množství operací, čili vyvolá množství událostí, v rámci jedné role. Ve chvíli, kdy roli změní, vyvolává opět nějakou dobu jen události dané role. Za celou dobu práce se systémem uživatel vystřídá minimum různých rolí, a spousta událostí v řadě je tedy provedena v rámci jedné role. Právě této skutečnosti lze využít pro zvýhodnění událostí při jejich vyhledávání. Je nejpravděpodobnější, že následující událost bude definovaná pro stejnou roli, pro kterou byla definovaná předchozí nalezená událost.

K upravení pořadí událostí je využita dříve zmíněná technologie LINQ. Následující příkaz setřídí množinu objektů z kolekce *udalosti* tak, že prvním rozhodujícím faktorem je minimální délka regulárního výrazu události, druhým je délka ve znacích daného výrazu a nakonec jsou díky číslu jedna upřednostněny ty události, které jsou definované pro stejnou roli jako poslední nalezená událost.

```
IEnumerable<UdalostClass> setrideno = from udalost in udalosti
   orderby udalost.Minimalni_delka descending,
      udalost.Reg_vyraz.Length descending,
      udalost.Role == rolePredchoziUdalost ? 1 : 2
   select udalost;
```

Jednotlivé části celé posloupnosti jsou v cyklu zpracovávány tak dlouho, dokud se velikost nezpracované části nerovná nule. Při zpracovávání se totiž z posloupnosti procedur v řetězcovém zápisu, odebírají jednotlivé zpracované procedury. Ve chvíli, kdy je zpracována poslední procedura, respektive je odebrán identifikátor poslední procedury, končí zpracování dané části a pokračuje se zpracováním další.

Zpracovávání části posloupnosti spočívá v samotném vyhledávání událostí. V cyklu je procházen seznam událostí a regulárním výrazům, které popisují dané události je předložen řetězec složený z identifikátorů procedur oddělených mezerami. Pokud je nalezena shoda regulárního výrazu s počátkem posloupnosti procedur, je vyhledávání ukončeno, a nalezená událost je vložena do kolekce dostupných událostí, které popisují danou posloupnost. Vhodným setříděním událostí je zajištěno to, že byla nalezena nejdelší možná část posloupnosti, která odpovídá některé události.

Dříve jsem uvedl, že stejná posloupnost procedur může odpovídat více událostem. Může se tak stát, že nalezená událost je jiná, než ta, která danou posloupnost procedur opravdu vyvolala. Z tohoto důvodu jsou vyhledány i další události, jejichž regulární výraz popisuje stejnou posloupnost, jakou rozpoznala první nalezená událost. Je důležité, že regulárním výrazům dalších vyhledávaných událostí není předložena stejná posloupnost, jako při vyhledávání první události, nýbrž jen ta část, kterou regulární výraz první události opravdu rozpoznal. Zápis regulárních výrazů totiž obsahuje i identifikátory procedur, které nemusí být uplatněny vždy, a tak se rozpoznaná část může případ od případu lišit. Aby bylo zajištěno, že při druhém vyhledávání události budou nalezeny jen ty, které zcela popisují předloženou posloupnost, jsou regulární výrazy událostí před použitím upraveny. Na

jejich začátek je umístěn symbol ‚^‘, který říká, že se shoda s regulárním výrazem musí nacházet na začátku předložené posloupnosti. Na konec výrazu je pak umístěn symbol ‚$‘, který naopak vyžaduje shodu výrazu s koncem posloupnosti. Všechny takto nalezené události jsou přidány do kolekce dostupných událostí.

Aby nebylo množství nalezených událostí, které popisují stejnou posloupnost, příliš mnoho, je seznam událostí zredukován jen na ty, které jsou definované pro stejnou roli, jako je role první nalezené události, nebo jako je role poslední úspěšně nalezené události[10]. Pro omezení množiny dat byla opět použita technologie LINQ.

```
IEnumerable<UdalostClass> dalsiUdalosti = from
   dalsiUdalost in udalosti
where dalsiUdalost.Role == udalost.Role
   || dalsiUdalost.Role == rolePredchoziUdalost
select dalsiUdalost;
```

Omezením množství alternativních událostí se značně sníží nároky na vyhledávání aktivit v posloupnosti událostí.

Jako platná událost je v množině dostupných událostí označena ta, která byla nalezena jako první. To ovšem neznamená, že je to správně. Budu vycházet z dříve uvedené myšlenky o posloupnosti událostí definované pro stejnou roli, z čehož vyplývá, že jako platná by měla být nastavena událost, která je definovaná pro stejnou roli jako předchozí platná událost. I přes to, že výběr událostí probíhal pouze z okruhu dvou rolí, může se stát, že tomu tak není. Proto je na závěr upravena platnost událostí tak, aby odpovídala výše uvedenému.

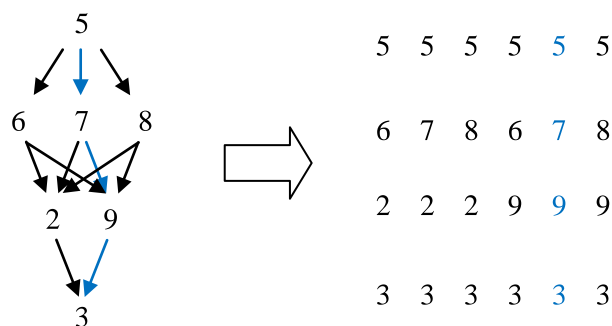
V této chvíli je vytvořen nový objekt PresentUdalostClass, který bude prezentovat nalezenou událost jako položka v GridView.

Pokud není při zpracovávání části celé posloupnosti nalezena žádná událost, je procedura z počátku posloupnosti odebrána a prohlášena za neidentifikovatelnou událost. Což ve výsledku značí, že daná procedura byla volána mimo známé posloupnosti a jde tedy nejspíše o pokus o napadení databáze.

Uvedený postup se opakuje pro všechny části posloupnosti až do doby, kdy je zpracována celá posloupnost daného uživatele. Výstupem metody zpracujLog je kolekce objektů PresentUdalostClass.

4.4.5 Nalezení aktivit

Vyhledávání aktivit v posloupnosti událostí je velmi podobné vyhledávání událostí. Ve vstupní posloupnosti se ovšem nenachází jen záznamy o jednom členu (jedné události), ale i takové, které mají členů *n*. Vznikají tedy různé varianty posloupnosti, z nichž jen jedna je ta správná. Situace je znázorněna na obrázku Obrázek 4.2 – Různé varianty posloupnosti událostí. Z posloupnosti čtyř událostí, z nichž druhá má dvě alternativní události a třetí má alternativu jednu, lze vytvořit šest různých variant posloupností, ve kterých je třeba nalézt sled aktivit.



Obrázek 4.2 – Různé varianty posloupnosti událostí

Ve chvíli, kdy alternativních událostí bude více, a posloupnost událostí bude delší, může nastat situace, kdy možných variant posloupností budou stovky nebo tisíce. Výběr správné varianty a nalezení v ní sledu aktivit, bude v takovém případě velmi výkonově náročné.

Množství alternativních událostí v této chvíli již omezit nelze. Lze však celou posloupnost událostí daného uživatele rozdělit na menší části a ty zpracovávat zvlášť. Údaje, podle kterých lze celou posloupnost rozdělit na menší části, tvoří množina událostí nulté úrovně, o kterých jsem se zmínil v kapitole 4.3.3.4. Ve chvíli, kdy uživatel vyvolá událost, která je dosažitelná v jakémkoliv okamžiku, dochází k započetí nové aktivity. Na základě tohoto faktu lze posloupnost událostí rozdělit na menší části, které lze rychleji a jednodušeji zpracovat.

Pro každou zpracovávanou část celé posloupnosti, je na základě alternativních událostí vygenerována množina variant posloupností. Každá varianta je pak dále zpracovávána a výsledek každé z nich je uložen v kolekci objektů VariantaPosloupnosti. Varianta posloupnosti je reprezentována řetězcem složeným z identifikátorů událostí oddělených mezerami. Při zpracovávání jsou z počátku řetězce odebírány patřičné identifikátory a zpracovávání končí ve chvíli, kdy jsou odebrány identifikátory všech událostí.

Vyhledávání aktivit opět probíhá na základě nalezení shody regulárního výrazu popisujícího aktivitu, s počátkem varianty posloupnosti. Vzhledem k tomu, že celá aktivita musí proběhnout v rámci jedné role, jsou aktivity vyhledávány pouze v seznamu aktivit, které jsou definovány pro stejnou roli, jakou má první událost dané varianty posloupnosti. Ve chvíli, kdy je nalezena aktivita odpovídající posloupnosti událostí, mohou nastat dvě situace. První je taková, že jde o aktivitu s originálním regulárním výrazem, tedy aktivita byla dokončena celá a vše je v pořádku. Druhá možnost je ta, že jde o aktivitu s upraveným regulárním výrazem, který je schopen rozpoznat i nedokončené aktivity. Ty mají ovšem ten neduh, že jejich regulární výrazy popisují stejnou posloupnost událostí jako kratší aktivity s originálním výrazem. Na výstupu by tak mohla místo dokončené kratší aktivity být vypsána nedokončená delší aktivita. Aby se tomuto předešlo, pokusím se nalézt aktivitu s originálním výrazem, který popisuje stejnou posloupnost událostí, jako první nalezená aktivita. Po rozhodnutí o správné aktivitě a zpracování celé varianty posloupnosti, je

[10] Tedy události, která byla nalezena v minulém vyhledávacím cyklu, a byla prohlášena za platnou.

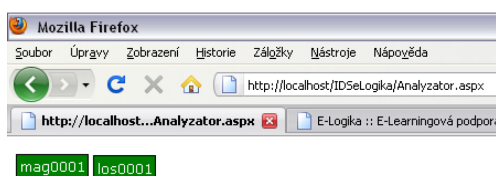
seznam nalezených aktivit uložen do množiny výsledků variant posloupností. Pokud by posloupnosti dané varianty neodpovídala žádná aktivita, bude odebrána událost z počátku varianty posloupnosti a následně bude označena jako neidentifikovatelná aktivita.

Pokud při zpracovávání variant dojde k situaci, že je některá varianta posloupnosti zpracována celá a bez neidentifikovatelných aktivit, je automaticky prohlášena za tu správnou a zpracovávání variant je zastaveno. V opačném případě jsou zpracovány všechny varianty a dle jejich výsledků je nalezena ta, která obsahuje nejméně neidentifikovatelných aktivit. Ta je pak prohlášena za správnou.

Ve chvíli, kdy jsem nalezl správnou aktivitu, čili správnou variantu posloupnosti událostí, můžu jí porovnat s aktuálně použitou variantou a podle změn upravit platnost jednotlivých událostí v celé posloupnosti. Podle výsledku analýzy posloupnosti událostí se tak upřesňuje výsledek analýzy posloupnosti procedur.

4.5 Ukázka výsledné aplikace

Celá analýza záznamů se spouští zobrazením stránky Analyzator.aspx ve webovém prohlížeči. Z auditních logů se načtou nové záznamy a spustí se analýza. Podle toho, jací uživatelé přistupovali k databázi, zobrazí se seznam jím náležících uživatelských jmen.



Obrázek 4.3 - Seznam uživatelů se záznamem v logu

Pokud je tlačítko s uživatelským jménem zelené, nevyskytuje se v auditních záznamech žádná nesrovnalost. Pokud by bylo tlačítko červené, značí to, že záznamy pro daného uživatele obsahují posloupnost procedur, která neodpovídá žádné události. Což je důvodné podezření na neoprávněný přístup k datům systému eLogika. Pokud má tlačítko zesílený spodní okraj, znamená to, že záznamy obsahují posloupnost událostí, která neodpovídá žádné aktivitě. To samo o sobě nemusí značit nic špatného. Pravděpodobně tato situace nastala v důsledku používání eLogiky ve více než jednom okně prohlížeče současně, případně v zadání konkrétní URL stránky do adresního řádku prohlížeče. Podle barvy tlačítek tak administrátor hned ví, zda je třeba se některým uživatelem detailněji zabývat, nebo je vše v pořádku a nemusí analýze dále věnovat pozornost.

Po zobrazení detailů některého z uživatelů, je zobrazen seznam aktivit, které byly identifikovány dle auditních záznamů.

mag0001 los0001

Login uživatele	Role	ID	Název aktivity	Popis aktivity	Čas spuštění
los0001	Tajemník	22	TA-zmena_roku-01	Změna akademického roku (v menu)	0:34:7.447
los0001	Tajemník	33	TA-seznam_kurzu-01	Zobrazení seznamu kurzů v daném rozložení ak. roku	0:34:9.761
los0001	Tajemník	43	TA-editace_uzivatele-01	Editace profilu uživatele	0:34:18.583
los0001	Tajemník	20	TA-detail_skoly-01	Zobrazení informací o škole	0:34:35.518

Obrázek 4.4 - Seznam aktivit konkrétního uživatele

Výpis aktivit obsahuje název role, v rámci které byla aktivita vykonána, čas, kdy byla vykonána a hlavně potom popis aktivity. Pokud není aktivita podbarvena žádnou barvou, tak je část auditních záznamů, které byly identifikovány jako daná aktivita, v pořádku. Pokud je aktivita podbarvena oranžovou barvou, tak to značí, že aktivita nebyla vykonána celá. Nejde tedy o situaci, kdy by byla ohrožena bezpečnost dat, jen je tak označena aktivita, která byla přerušena před svým dokončením. Pokud by byla aktivita podbarvena červenou barvou, jde o ‚Neznámou aktivitu'. Tedy posloupnost událostí, která neodpovídá žádné aktivitě.

U každé aktivity si lze dále zobrazit její detaily. Stačí na ní kliknout myší a zobrazí se tabulka se seznamem událostí, které danou aktivitu tvoří.

mag0001 los0001

Login uživatele	Role	ID	Název aktivity	Popis aktivity	Čas spuštění
los0001	Tajemník	22	TA-zmena_roku-01	Změna akademického roku (v menu)	0:34:7.447
los0001	Tajemník	33	TA-seznam_kurzu-01	Zobrazení seznamu kurzů v daném rozložení ak. roku	0:34:9.761
los0001	Tajemník	43	TA-editace_uzivatele-01	Editace profilu uživatele	0:34:18.583

ID události	Počet variant	Popis události	Posloupnost procedur
52	7	Správa uživatelů - zobrazení formuláře	1 1 36 33 33 33 13 12
69	1	Hlédání uživatelů	1 1 40 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
70	2	Načtení detailů uživatele k editaci	1 21 1 36 33 33 33 13 12
282	3	Uložení upravených údajů uživatele	1 6 1 36 33 33 33 13 12

Vrací seznam rolí pro daného uživatele v dané škole

Login uživatele	Role	ID	Název aktivity	Popis aktivity	Čas spuštění
los0001	Tajemník	20	TA-detail_skoly-01	Zobrazení informací o škole	0:34:35.518
los0001	Tajemník	20	TA-detail_skoly-01	Zobrazení informací o škole	0:43:12.890

Obrázek 4.5 - Detaily aktivity

Detailní informace o aktivitě obsahují hlavně popis jednotlivých událostí a dále seznam procedur, které dané události tvoří. Respektive posloupnost procedur, která byla identifikována jako daná událost. Vzhledem k tomu, že daná posloupnost procedur může být velmi dlouhá, rozhodl jsem se vypisovat ji jen pomocí identifikátorů jednotlivých procedur. Jakákoliv jiná forma výpisu by byla příliš prostorově náročná. Na informační hodnotě se však nic neztratilo, jelikož popis, co která procedura dělá s daty v databázi, se zobrazí ve chvíli, kdy nad jejím identifikátorem zastaví kurzor myši.

Nyní se pokusím nastínit, jak lze z výstupu analýzy auditních záznamů rozeznat neoprávněný přístup k databázi. Nejprve uvedu příklad na nestandardní pohyb uživatele v rámci aplikace.

Dejme tomu, že uživatel běžně pracuje se systémem eLogika. Zrovna se nachází na stránce s výpisem uživatelů dané školy a do adresního řádku prohlížeče vloží URL /Pages/Zakladni/KurzDetail/KurzQuest.aspx, na které se nachází seznam kurzů v konkrétním semestru vybraného akademického roku. Výstup analýzy bude vypadat takto:

los0001	Tajemník	33	TA-seznam_kurzu-01	Zobrazení seznamu kurzů v daném rozložení ak. roku	2:19:22.43
los0001	Tajemník	41	TA-seznam_uzivatelu-01	Zobrazení seznamu uživatelů	2:19:38.396
los0001		0	Neznámá aktivita	Daná posloupnost událostí neodpovídá žádné aktivitě	

ID události	Počet variant	Popis události	Posloupnost procedur
299	1	Seznam kurzů v rozdělení akademického roku	34 1 36 33 33 33 13 12

los0001	Tajemník	20	TA-detail_skoly-01	Zobrazení informací o škole	2:20:10.191
los0001	Tajemník	33	TA-seznam_kurzu-01	Zobrazení seznamu kurzů v daném rozložení ak. roku	2:21:13.919

Obrázek 4.6 - Výpis s nestandardním chováním uživatele v aplikaci

Aktivita, která odpovídá posloupnosti událostí začínající událostí s identifikátorem 299 (tu následuje posloupnost událostí 47 59 93 93, kvůli velikosti obrázku jsem ovšem nezobrazil detaily aktivit, které jsou těmito událostmi tvořené), ovšem neexistuje a tak jí systém identifikoval jako ‚Neznámá aktivita'. Při zobrazení detailů dané aktivity je možné vidět, že ji tvoří konkrétní existující událost.

Pokud by se útočník dokázal připojit k databázovému serveru a mohl si spustit jakoukoliv proceduru či manipulovat přímo s daty v tabulkách, na výstupu analýzy by se to projevilo červeným tlačítkem uživatele, s tučným spodním okrajem. Já pro následující ukázku použil účet sa databázového serveru, a prostřednictvím SQL Server Management Studia jsem spustil proceduru garant_find, která slouží k vyhledání uživatelů v roli garant, dle zadaného jména.

mag0001 los0001 dbo

Login uživatele	Role	ID	Název aktivity	Popis aktivity	Čas spuštění
dbo		0	Neznámá aktivita	Daná posloupnost událostí neodpovídá žádné aktivitě	

ID události	Počet variant	Popis události	Posloupnost procedur
0	1	Procedury nezapadají do žádné posloupnosti	147 143 143 143 143 143 143 143

Vrací seznam garantů odpovídající danému jménu

Obrázek 4.7 - Vástup analýzy s nalezeným neoprávněným přístiupem k datům

Vzhledem k tomu, že samotné zavolání dané procedury neodpovídá žádné aktivitě, systém danou posloupnost označil za neznámou aktivitu. Současně však neexistuje žádná událost, která by dané posloupnosti odpovídala a tak byla rovněž označena za neznámou. Výpis posloupnosti procedur pak informuje o tom, že byla zavolána procedura vracející seznam garantů. Identifikátory 143 v posloupnosti znamenají volání příkazu SELECT na některou tabulku. Ty jsou zde zobrazeny proto, že dle nastavení auditu se mají logovat pro všechny uživatele, kteří nejsou v některé databázové roli systému eLogika, jak volání procedur, tak i jednotlivých příkazů SELECT, UPDATE, INSERT, DELETE. A jak jsem zmínil dříve, tak do auditu jsou zahrnuty i sledované operace spuštěné v rámci volání jiných příkazů. Zde to bylo opakované spuštění příkazu SELECT (výběr z několika tabulek) v rámci volání procedury.

4.5.1 Dostupnost ukázkové aplikace

Spuštění celého systému není zcela triviální, a tak jsem připravil dvě varianty ukázkové aplikace, které lze spustit a prohlédnout si systém za běhu. Obě vyžadují provoz na OS Microsoft Windows XP SP 2, či novějším.

4.5.1.1 Odlehčená varianta aplikace

První z nich je odlehčená varianta, obsahující pouze samotný systém pro detekci neoprávněné manipulace s daty. Tato varianta nekomunikuje se systémem eLogika a pro analýzu využívá pouze záznamy, které již má shromážděny ve své databázi. Pro spuštění je třeba mít nainstalován databázový systém Microsoft SQL Server 2008 v libovolné verzi s povoleným způsobem přihlášení Windows Authentications. Dále pak vývojové prostředí Microsoft Visual Studio 2008 (aplikaci lze spustit i ve webovém serveru IIS, ale tato varianta by byla příliš složitá na nastavení). Aby bylo možné aplikaci spustit, je třeba provést následující kroky.

- Na přiloženém CD se v adresáři ‚System detekce napadeni DB ' nachází SQL soubor pro

vytvoření potřebné databáze. Ten je potřeba spustit. Automaticky se spustí vytvoření databáze a naplnění daty.

- V adresáři ‚System detekce napadeni DB\IDSeLogika‘ se nachází soubor ‚IDSeLogika.sln‘, ten automatický spouští projekt ve Visual Studiu. Po té co se projekt otevře, stačí stisknout kombinaci kláves Ctrl+F5, která spustí samotnou webovou aplikaci ve webovém prohlížeči.

4.5.1.2 Plná varianta aplikace

Plná varianta počítá se spoluprácí se systémem eLogika. Aby tato varianta byla plně funkční, je potřeba mít nainstalovánu Enterprise verzi Microsoft SQL Serveru, s povoleným způsobem přihlášení ‚SQL Server Authentication‘. Kroků, které se musí dále provést je znatelně více, než v odlehčené variantě.

- Nejprve je potřeba vykonat všechny kroky, které jsou popsány pro spuštění odlehčené varianty.
- Je třeba vytvořit adresářovou strukturu ‚c:\elogika\audit\‘ na systémovém disku
- Pod uživatelem sa (administrátor SQL serveru) je potřeba spustit následující skripty v tomto pořadí:
  - Z adresáře ‚SQL Skripty‘ skript krok1.sql
  - Z adresáře ‚eLogika‘ skript CreateDBeLogika.sql, který vytvoří databázi pro systém eLogika
  - Z adresáře ‚SQL Skripty‘ skript krok2.sql
  - Z adresáře ‚SQL Skripty‘ skript krok3.sql
  - Z adresáře ‚SQL Skripty‘ skript krok4.sql
- Z nabídky start vybrat položku ‚spustit‘ a do pole ‚Otevřit‘ zadat ‚regedit‘
  - V klíči ‚HKEY_LOCAL_MACHINE/SOFTWARE‘ je potřeba vytvořit nový klíč s názvem ‚eLogika‘ (v kontextové nabídce klíče SOFTWARE vybrat položku Nový/Klíč)
  - V klíči elogika pak vytvořit klíč ‚connStr‘
- V adresáři ‚eLogika\E-Logika‘ se nachází soubor ‚E-Logika.sln‘ který otevře projekt systému eLogika ve Visual Studiu. Kombinací kláves Ctrl+F5 se opět provede spuštění aplikace.
- Do zobrazeného přihlašovacího formuláře lze zadat jméno ‚admin‘ s heslem ‚admin‘, což způsobí přihlášení jako systémový administrátor eLogiky. V této roli lze vytvářet další uživatele.
- Aby se projevily změny v systému pro odhalování neoprávněného přístupu k datům, je potřeba před spuštěním analýzy navštívit stránku ‚Update.aspx‘ v prohlížeči, kde je systém spuštěn. Po aktualizaci auditních záznamů lze na stránce ‚Analyzator.aspx‘ jejím obnovením spustit novou analýzu.

5 Bezpečnost webových aplikací

V předchozích kapitolách byl popsán systém, který kontroluje přístup k datům uloženým v databázi. Tato kontrola je závislá na datech, které jsou získány na straně serveru a lze tak odhalit i útoky, které nebyly provedeny prostřednictvím systému eLogika.

Na bezpečnost je ovšem třeba pohlížet dvěma různými způsoby. Prvním z nich je dopad, jaký může mít ztráta dat. V případě e-learningového systému je jedna z nejkritičtějších částí databáze ta, která obsahuje otázky a odpovědi, sloužící k ověřování znalostí studentů. Únik takovýchto dat může způsobit neoprávněné dosažení lepších výsledků, což v konečném důsledku může vést i k úspěšnému absolvování kurzu, na které nemá student dostatečné znalosti. Pokud už k takovémuto úniku dat dojde, je důležité, aby se to dověděly kompetentní osoby a mohly tak provést patřičnou nápravu. V tomto případě by šlo hlavně o anulování testů provedených po odcizení dat. Takovýto přístup řeší, v rámci eLogiky, dříve popsaný systém na odhalování neoprávněné manipulace s daty.

Druhým způsobem jak na bezpečnost nahlížet, je samotná ochrana dat, kterými aplikace disponuje, před jejich zneužitím. Tu zajišťuje jednak správné řízení přístupu k datům, ale také prevence proti útokům, které lze na webové aplikace provést.

Převážná většina útoků na webové aplikace, je prováděna skrze bezpečnostní díry v jejich webovém rozhraní. Ty vznikají vinou vývojářů, kteří si neuvědomují možné následky napadení aplikace, případně neznají možné typy útoků a způsoby ochrany proti nim.

5.1 Nejčastější útoky na webové aplikace

V následujících odstavcích jsou uvedeny nejčastější útoky, se kterými se dnes webové aplikace potýkají.

5.1.1 SQL Injection

Jde o útok na databázový server prostřednictvím modifikovaných SQL dotazů (15). Cílem útoku může být získání přístupu k webové aplikaci, odcizení dat, jejich modifikace či úmysl čistě destruktivní, což se rovná smazání dat, tabulek nebo celé databáze.

Princip útoku je postaven na dynamickém sestavování SQL dotazů, posílaných na databázový server, na základě požadavků uživatele aplikace. Základní předpoklad pro úspěšný útok, je vytváření SQL dotazů jednoduchým skládáním řetězců. V takovém případě je uživatelem zadaná informace vložena do SQL dotazu a ten je následně odeslán k vykonání na databázový server. Pokud uživatel zadá řetězec, který má v jazyce SQL určitý význam, může dojít k modifikaci původního SQL dotazu, jehož provedení bude mít jiný, než původně zamýšlený výsledek.

5.1.1.1 Příklad útoku

První možnost k použití SQL Injection, má útočních již při přihlášení k aplikaci. To většinou vyžaduje vyplnění uživatelského jména a hesla. Při skládání řetězců by SQL dotaz mohl vypadat následovně:

```
SELECT * FROM users WHERE login = 'jmeno' AND password = 'heslo'
```

Úspěšné přihlášení k aplikaci proběhne ve chvíli, kdy je vrácen nenulový výsledek dotazu. Pokud

by ovšem útočník zadal do vstupního pole pro uživatelské jméno řetězec: `' OR 1=1 --`, výsledný dotaz by byl:

```
SELECT * FROM users WHERE login = '' OR 1=1 --' AND password =
'heslo'
```

Vzhledem k tomu, že podmínka „`login = '' OR 1=1`“ je vždy pravdivá a dvě pomlčky uvozují v jazyce SQL komentář, vrátí dotaz vždy nenulovou množinu záznamů, čímž dojde k přihlášení útočníka k aplikaci. Dotaz se dá také upravit tak, že vložíme existující uživatelské jméno a výše zmíněný řetězec vložíme do pole pro heslo. Aplikace tak útočníka přihlásí pod konkrétním účtem.

Pokud je aplikace zranitelná na SQL Injection, může útočník, v závislosti na množině oprávnění, které má aplikace na databázovém serveru přidělena, provádět prakticky jakékoliv operace. Například neškodný dotaz na výpis informací o městě na základě jeho názvu, lze zneužít ke smazání celé tabulky. Stačí místo názvu města zadat řetězec: „`' ; DROP TABLE mesto --`“, který upraví SQL dotaz na:

```
SELECT * FROM mesto WHERE nazev = '' ; DROP TABLE mesto --'
```

Na databázový server jsou tedy odeslány dva dotazy, kde první je ukončen středníkem, a druhým dotazem je provedeno smazání celé tabulky mesto.

5.1.1.2 Ochrana proti SQL Injection

Základní ochranou proti modifikaci SQL dotazů je důsledná kontrola hodnot, které jsou do SQL dotazů vkládány. Starší metody ochrany jsou založeny na nahrazování apostrofů uvozovkami, případně escapování nebezpečných znaků zpětným lomítkem.

V dnešní době, kdy jsou k vývoji webových aplikací používány nejrůznější frameworky, je situace mnohem jednodušší. Ty totiž zpravidla obsahují vlastní databázovou vrstvu, která využívá parametrizovaných dotazů. To znamená, že na místo hodnot, které chceme do SQL dotazu vložit, přidáme do dotazu zástupné řetězce (parametry), které později naplníme požadovanými hodnotami. O ošetření hodnot parametrů se na základě jejich datového typu stará samotná databázová vrstva. Je tedy vyloučeno, že by vývojář některou z hodnot, vstupující do SQL dotazu, ošetřil nedokonale či vůbec.

Systém eLogika využívá ještě o kousek bezpečnější metodu, která je založena na parametrizovaných procedurách. Požadavky na manipulaci s daty nejsou z aplikace odesílány pomocí klasických SQL dotazů, ale využívají se procedury uložené na databázovém serveru, které jsou z aplikace volány pouze na základě svého názvu. Dochází tak k oddělení aplikace od celé struktury databáze. Jaké tabulky jsou pro získání konkrétní množiny dat použity, lze zjistit pouze nahlédnutím do těla procedury, což ovšem není z aplikace dovoleno. Samozřejmostí je vkládání vstupních parametrů procedur odděleně, pomocí výše zmíněných zástupných řetězců. Ty musí nyní odpovídat jednotlivým vstupním parametrům procedury.

Například hlavička procedury pro smazání studenta ze systému definuje tři vstupní parametry:

```
CREATE PROCEDURE [dbo].[student_delete]
(
      @id_uzivatel      int,
      @id_trida         int,
      @id_skola         int
)
```

Korektní volání procedury z aplikace pak bude vypadat takto:

```
SqlConnection conn = new SqlConnection(connectionString);
SqlCommand cmd = new SqlCommand("student_delete", conn);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add(new SqlParameter("@id_uzivatel", id_u));
cmd.Parameters.Add(new SqlParameter("@id_trida", id_t));
cmd.Parameters.Add(new SqlParameter("@id_skola", id_s));
```

5.1.2 Cross Site Scripting (XSS)

Tento typ útoku využívá špatného ošetření vstupních dat, která se vypisují zpět do webových stránek (16). Základem útoku je nalezení místa, kudy lze do aplikace vložit kus kódu napsaného v klientském skriptovacím jazyku. To by samo o sobě nebylo nebezpečné, pokud by se daný kód nevypisoval zpět do webové stránky. Daný kód totiž není vykreslen jako jiné prvky stránky, ale spuštěn v prohlížeči. Možnosti, jakým lze škodlivý kód dostat do obsahu stránek aplikace, jsou v podstatě dva. První, který je zcela zřejmý, je využití některého formuláře pro vkládání dat, které jsou opětovně zobrazeny na stránce. Druhý způsob využívá situace, kdy jsou informace mezi stránkami předávány pomocí parametrů v URL adrese, přičemž jsou hodnoty parametrů následně vkládány do obsahu stránek.

Díky širokým možnostem skriptovacích jazyků na straně klienta, nejčastěji pak JavaScriptu, lze pomocí tohoto útoku dosáhnout kompletní změny vzhledu webové stránky, odcizení cookies uživatele, sledování stisků kláves nebo přetížení serveru pomocí opakovaných požadavků na zobrazení stránky.

Na základě toho, zda je nebezpečný kód pouze jednorázově přepsán zpět do stránky, nebo je uložen v databázi (případně v jiném datovém úložišti) a je tedy zobrazován opakovaně, rozlišujeme okamžité a trvalé XSS útoky. Okamžité útoky využívají upravených odkazů, které obsahují škodlivý kód, který je následně vložen do obsahu webové stránky, kde je spuštěn. Aby byl útok úspěšný, je zapotřebí ‚spolupráce' uživatele, který musí na daný odkaz navštívit. Trvalé útoky naopak využívají toho, že vstup, který útočník zadá, se vypisuje v nezměněné podobě ostatním návštěvníkům dané stránky. Tento typ útoku je velmi nebezpečný právě kvůli tomu, že může postihnout obrovské množství nic netušících uživatelů.

5.1.2.1 Ochrana proti XSS

Ochranou proti XSS je opět důkladná kontrola vstupních, ale i výstupních dat. Pokud nepovolíme vložení nebezpečného kódu, odstraníme tak největší riziko útoku. Je však možné, že se útočníkovi podaří kód do aplikace vložit jinudy, než pomocí samotné aplikace. Například přímým přístupem k databázi. V takovém případě není kontrola vstupů nic platná a od úspěšného útoku může ochránit pouze kontrola výstupu, potažmo jeho úprava.

Ve výchozím nastavení aplikací ASP.NET je zapnuta kontrola všech vstupů na přítomnost potencionálně nebezpečného kódu. Pokud se tak útočník pokusí například do formuláře, či parametru URL vložit JavaScriptový kód „`<script>alert("XSS útok !")</script>`“, bude vyvolána výjimka HttpRequestValidationException a zpracování požadavku bude přerušeno, případně bude vyvolána obsluha uvedené výjimky. Pokud chceme tuto kontrolu vstupů pro konkrétní stránku vypnout, je potřeba upravit nastavení dané stránky změnou hodnoty atributu validateRequest na false.

```
<%@ Page validateRequest="false" %>
```

Pokud chceme kontrolu vstupů vypnout globálně pro celou aplikaci, lze tak učinit z konfiguračního souboru web.config

```
<configuration>
   <system.web>
      <pages validateRequest="false" />
   </system.web>
</configuration>
```

V rámci eLogiky je kontrola vstupů přenechána technologii ASP.NET, což je zajištěno nastavením vlastnosti ValidateRequest na hodnotu true. Vyvolaná výjimka, při případném závadném vstupu, je odchycena funkcí Application_Error, která je definovaná v konfiguračním souboru global.asax. Funkce Application_Error přesměruje uživatele, na základě chyby, která jí vyvolala, na stránku s informacemi o vzniklé situaci.

Pro ošetření výstupu, tedy míst, kde dochází k vypisování dat z databáze, je vhodné použít metodu HtmlEncode z jmenného prostoru Server. Ta převede všechny potencionálně nebezpečné znaky na HTML entity, které se při vykreslování stránky přepíšou zpět na původní znaky. Stejnou funkci, pouze pro výpis dat předaných pomocí URL parametrů, zastává metoda UrlEncode ze stejného jmenného prostoru. Obě tyto metody jsou použity pro ošetření dat, která se vypisují do stránek v systému eLogika.

5.1.3 Cross Site Request Forgery (CSRF)

Smyslem tohoto útoku je provedení nějaké operace v rámci webové aplikace, na jejíž provedení má uživatel oprávnění, avšak není aktuálně v jeho zájmu její provedení uskutečnit. Jde tedy o zneužití oprávnění uživatele k provedení operace bez jeho vědomí (17).

Nejčastěji je k provedení operace využit sám uživatel, kterému je předložen upravený odkaz, který spouští danou operaci. Uživatel je pak nalákán na navštívení daného odkazu, čímž operaci sám vyvolá. Jako jednoduchý příklad může sloužit hlasování v anketě. Běžný postup hlasování je ten, že si uživatel zobrazí anketu a výběrem jedné z možností provede hlasování. U CSRF útoku podstrčí útočník uživateli vlastní odkaz, který je upravený tak, aby automaticky provedl hlasování pro jednu z možností. Po té, co útočník ‚přinutí' uživatele odkaz navštívit, je provedeno hlasování a uživateli je jen zobrazena stránka s výsledky hlasování. To však může vést ke snadnému odhalení útoku. Útočník může uživateli zobrazit obsah, na který ho nalákal (kvůli kterému uživatel navštívil odkaz) a zmíněné hlasování v anketě provést automaticky pomocí JavaScriptu ve skrytém rámu stránky, kterou uživatel navštívil. Hlasování tedy může být provedeno i z jiné stránky a stačí jen, aby byl uživatel přihlášen k webu, kde útočník zamýšlí hlasovat. Proti této variantě existuje však snadná ochrana, kdy je před započtením hlasu ověřena hodnota referer, která obsahuje informaci o tom, odkud (z jaké webové adresy) požadavek přišel. Pokud se původce požadavku neshoduje s adresou webu, kde je umístěna anketa, hlas nebude započítán.

Celý útok lze vylepšit o eliminaci účasti uživatele. Stačí, pokud se útočníkovi podaří do stránky vložit některý z HTML tagů, který umožňuje načtení zdroje dat. Je to například „img" pro vložení obrázku, „applet" pro vložení Java Appletu, „iframe" pro plovoucí rám a další. Pak stačí do atributu, který definuje cestu k datům (běžně je to „src") vložit adresu pro hlasování v anketě, a každý uživatel, který si zobrazí stránku obsahující útočníkem vložený HTML tag, provede

hlasování v anketě, aniž by o tom vůbec věděl.

Možností zneužití pomocí CSRF jsou mnohem širší, než pouhé hlasování v anketě. V podstatě jde tento útok použít kdekoliv, kde jsou odesílána data k dalšímu zpracování.

5.1.3.1 Ochrana proti CSRF

Jednou z možností je kontrola hodnoty referer, jak jsem zmínil výše. Bohužel, o odesílání této hodnoty se stará prohlížeč a ten může tuto hodnotu blokovat, nebo upravovat. Ochrana tedy nespočívá v samotné aplikaci a tak jí nelze považovat za obecně použitelnou.

Za nejúčinnější automatickou ochranu je považováno ověřování takzvaných tokenů. Jsou to jedinečné, náhodné identifikátory, které jsou generovány pro všechny formuláře v rámci aplikace. Při načtení stránky s formulářem je vygenerován token, který je pro danou akci a daného uživatele uložen na serveru a současně je buďto přidán do skrytého pole formuláře, nebo se stane součástí URL, kam bude odeslán požadavek na zpracování formuláře. Ve chvíli, kdy je požadavek na operaci odeslán, je doručený token porovnán s tokenem uloženým na serveru a v případě shody je operace provedena. Pokud se tokeny neshodují, operace je zamítnuta. V obou případech je token ze serveru odstraněn, aby nemohl být znovu použit.

Tato metoda ochrany selhává ve chvíli, kdy je aplikace náchylná na napadení útokem XSS.
V takovém případě si může útočník nechat vygenerovat token bez vědomí uživatele a následně jej použít k ověření operace. Kritické operace je proto doporučeno podmínit opakovaným ověřením uživatele pomocí jeho hesla.

Metodu pro ověřování tokenu před provedením operace lze implementovat vlastními silami, což je výhodné ve chvíli, kdy se s vývojem celé webové aplikace začíná. Pokud je potřeba zabudovat ochranu před CSRF do již hotové aplikace, lze použít některý z modulů, který je pro používanou technologii dostupný. Pro ASP.NET je to modul AntiCSRF (18).

6 Závěr

Webové portály, aplikace, ale i obyčejné dynamické weby, jsou od samého počátku velkým lákadlem pro útočníky, kteří se snaží nalézt chybu v zabezpečení a tu následně využít pro napadení celé aplikace. V dnešním světě, kde se obchoduje s osobními údaji uživatelů, ale i s jinými odcizenými daty, hraje bezpečnost aplikací jednu z hlavních rolí.

Aby byla zajištěna dostatečná bezpečnost dat, je důležité vhodně řídit přístup k databázovému serveru a také ke konkrétním datům. Tato práce popisuje způsoby, kterými lze k datům přistoupit a popisuje způsob, jakým lze správu přístupu řídit. Krom správy přístupu k datům, obsahuje text i popis dalších metod a technologií, kterými lze zvýšit bezpečnost dat uložených v databázích systému Microsoft SQL Server 2008. Práce rovněž obsahuje ukázky z implementace popsaných metod, podle kterých si čtenář může zabezpečit vlastní databázový server.

Jedním z cílů práce byla implementace systému pro odhalování neoprávněného přístupu k datům. Ten není založen na kontrole zdroje požadavku, či jeho oprávněnosti, což jsou údaje, které se dají podvrhnout, ale na kontrole toho, jaká data a v jakém pořadí jsou požadována. Z těchto informací lze posoudit, zda požadavky odpovídají běžné práci v aplikaci, nebo jde o napadení databáze. Aby mohl být tento systém ošálen, musel by mít útočník přístup ke kódu sledované aplikace, což je velmi nepravděpodobné. Správná funkčnost systému je zcela závislá na datech, které popisují události e-learningovém systému eLogika. Vzhledem k tomu že eLogika stále prochází vývojem, je udržování systému pro odhalování neoprávněných přístupu k datům velmi náročné. Při dalším vývoji tohoto systému by se tak dalo zaměřit na návrh a implementaci modulu, který by automatizovaně kontroloval změny v eLogice a upravoval tak data v databázi systému.

Další vývoj by se také mohl zaměřit na automatizování celé kontroly. Nyní je třeba spustit kontrolu ručně, ale systém by se dal přepracovat tak, aby se kontrola prováděla v reálném čase. Po té co by se vyskytlo podezření na napadení databáze, mohl by být upozorněn administrátor a ten by mohl rychleji zasáhnout.

Použitá literatura

1. **Wikipedie, Přispěvatelé.** Learning Management System. *Wikipedie: Otevřená encyklopedie.* [Online] 27. Duben 2010. http://cs.wikipedia.org/wiki/Learning_Management_System.

2. —. E-learning. *Wikipedie: Otevřená encyklopedie.* [Online] 30. Duben 2010. http://cs.wikipedia.org/wiki/E-learning.

3. Connection strings for SQL Server 2008. *ConnectionStrings.com.* [Online] http://www.connectionstrings.com/sql-server-2008.

4. **Microsoft.** Informace o registru systému Windows pro pokročilé uživatele. *Technická podpora Microsoft.* [Online] 12. 3 2008. http://support.microsoft.com/kb/256986/cs.

5. —. ASP.NET IIS Registration Tool (Aspnet_regiis.exe). *MSDN Library.* [Online] 19. Únor 2009. http://msdn.microsoft.com/en-us/library/k6h9cz8h(VS.80).aspx.

6. **Šeda, Jan.** *DOTNET in Samples.* [Online] 2004. http://skilldrive.com/book/DOTNETinSamples.htm.

7. *CoderSource.net.* [Online] 2005. http://www.codersource.net/microsoft-net/c-basics-tutorials/working-with-system-registry.aspx.

8. **Brewer, William.** SQL Server Endpoints: Soup to Nuts. [Online] 6. Červenec 2007. http://www.simple-talk.com/sql/database-administration/sql-server-endpoints-soup-to-nuts/.

9. **Walters, Robert E.** *Mistrovství v Microsoft SQL Server 2008 : [kompletní průvodce databázového experta].* [překl.] Pavel Poloncý. Brno : Computer Press, 2009. str. 864. 978-80-251-2329-4 (váz.).

10. **Lee, Il-Sung a Rask, Art.** Microsoft SQL Server 2008 Auditing. *MSDN Library.* [Online] 9. Březen 2009. http://msdn.microsoft.com/en-us/library/dd392015.aspx.

11. **Microsoft.** fn_get_audit_file (Transact-SQL). *Microsoft TechNet.* [Online] Listopad 2009. http://technet.microsoft.com/en-us/library/cc280765.aspx.

12. **Hotek, Mike.** *Microsoft SQL Server 2008 - Implementation and Maintenance.* Redmond, Washington : Microsoft Press, 2009. 2008940530.

13. **Macrosoft.** SQL Server Audit Action Groups and Actions. *Microsoft TechNet.* [Online] Listopad 2009. http://technet.microsoft.com/en-us/library/cc280663.aspx.

14. **Wikipedie, Přispěvatelé.** LINQ. *Wikipedie: Otevřená encyklopedie.* [Online] 11. Listopad 2009. http://cs.wikipedia.org/wiki/LINQ.

15. SQL Injection (Full Paper). *Security-portal.cz.* [Online] 5. Listipad 2009. http://www.security-portal.cz/clanky/sql-injection-full-paper.

16. XSS (Cross-Site Scripting) hacking. *Security-portal.cz.* [Online] 17. Únor 2008. http://www.security-portal.cz/clanky/xss-cross-site-scripting-hacking.

17. Cross Site Request Forgery. *SOOM.cz.* [Online] 19. Květen 2008. http://www.soom.cz/index.php?name=articles/show&aid=484.

18. AntiCSRF - A Cross Site Request Forgery (CSRF) module for ASP.NET. *CodePlex.* [Online] 15. Prosinec 2008. http://anticsrf.codeplex.com/.

19. **Newman, Aaron C.** *Security Auditing In Microsoft SQL Server (White Paper).* místo neznámé : Application Security, INC.