Informatyka, studia dzienne semestr III

Pracownia Problemowa **2014**

Prowadzący: dr. inż Aneta Poniszewska-Marańda poniedziałek, 14:15

Data oddania: _____ Ocena: _____

Mateusz Grotek	186816
Mateusz Jakóbczak	186819
Rafał Jurkiewicz	186822
Łukasz Kotyński	186829
Paweł Tarasiuk	186875

Kontrola dostępu w dynamicznych systemach informatycznych

1. Problem

Przedstawiony problem dotyczy tworzenia i analizy zaawansowanych, inteligentnych mechanizmów kontroli dostępu w dynamicznych systemach informatycznych. Celem projektu jest zapoznanie się ze stosowanymi rozwiązaniami, zdobycie aktualnej wiedzy na ten temat, a także wykonanie praktycznej implementacji wybranych rozwiązań.

2. Wprowadzenie

Pierwszą rzeczą, jaką należy zrobić w celu rozwiązania postawionego werbalnie problemu jest dokładne zrozumienie określeń użytych w jego specyfikacji. W wypadku kontroli dostępu w dynamicznych systemach informatycznych istotną rzeczą jest zwrócenie uwagi na pojęcia kontroli dostępu i dynamicznego systemu informatycznego. Jako, że główna część zagadnienia skupia się na zapewnieniu kontroli dostępu, wyjaśnimy najpierw drugie, mniej istotne pojęcie, jakim są dynamiczne systemy informatyczne.

Pojęcie dynamiki można tutaj rozumieć na różne sposoby, ale na pewno podstawowym aspektem jest tutaj wrażliwość na zmieniające się warunki,

takie jak na przykład czas, lub pewne zdarzenia zachodzące zarówno w samym systemie, jak i poza nim. Oznacza to, że system nie powiniem robić żadnych założeń na temat tego co się aktualnie dzieje, a raczej reagować dynamicznie. Dla porównania możemy wskazać na przykład dostęp do plików na serwerze. Rozwiązanie statyczne mogłoby polegać na przykład na tym, że w systemie jest na stałe zapamiętane (zaszyfrowane) hasło, które użytkownik musi podać przy logowaniu, po którym uzyskuje pełny i nieograniczony czasowo dostęp do pewnych, na sztywno zdefiniowanych zasobów, takich jak pewne pliki w pewnych katalogach. Skontrastujmy to z systemem w którym dostęp do plików jest ograniczony pod względem na przykład ilości osób, które jednocześnie z nich korzystają, przy czym czas korzystania może być ograniczony w zależności od konkretnej osoby, a także od ilości osób. Poza tym dostęp może być włączany konkretnym osobom w konkretnych godzinach. Co więcej, uzyskanie dostępu może zależeć od wydania zezwolenia. Jak widać stworzenie systemu dynamicznego jest dużo trudniejsze od stworzenia systemu statycznego.

Wróćmy teraz do pojęcia kontroli dostępu. Kontrola dostępu oznacza, że dostęp do pewnych zasobów jest, z jednej strony, dozwolony lub zabroniony, natomiast z drugiej, że jest on nadzorowany, czyli że na przykład każde skorzystanie z zasobów jest zapisywane, lub nawet zatwierdzane przez odpowiednią osobę. Wynika z tego, że elementami kontroli dostępu są nie tylko autentykacja i autoryzacja, ale także wydawanie zgody na dostęp i audyt.

3. Aktualny stan wiedzy

Jest wiele różnych sposobów na zapewnienie kontroli dostępu. Co więcej sposoby te niekoniecznie kolidują ze sobą, a często mogą ze sobą współpracować. Do podstawowych mechanizmów mogących zapewnić kontrolę dostępu należą [1] [2] [3]:
— kontrola dostępu uznaniowa *Discretionary Access Control, DAC*
— kontrola dostępu obowiązkowa *Mandatory Access Control, MAC*
— kontrola dostępu oparta na rolach *Role Based Access Control, RBAC.*
Każdy z tych modeli różni się elementem, który przydziela prawa dostępu, a także mechanizmami dostępu [3].

W przypadku **DAC** zarządzaniem uprawnieniami zajmują się właściciele danych. Często wykorzystywane są do tego listy kontroli dostępu (*Access Control Lists, ACLs*). Dobrym przykładem takiego rozwiązania jest dostęp do plików i katalogów na dysku w Uniksie/Linuksie. Właściciel pliku/katalogu może ustawić prawa dostępu dla siebie, grupy, do której należy dany plik, a także dla osób spoza grupy. Prawa te, to prawa odczytu, zapisu i wykonywania, a także pewne dodatkowe prawa o których nie będziemy tutaj pisać. Jednoczesną zaletą i wadą takiego rozwiązania jest fakt, że zarządzanie uprawnieniami dokonywane jest przez właścicieli zasobów. Jest to zaleta, gdyż łatwo jest zarządzać właścicielowi zasobami, które tylko on posiada, gdyż ich ilość jest ograniczona. Jest to wada, gdyż brak jest centralnego nadzoru nad uprawnieniami.

Przypadek **MAC** zakłada, że przydzielaniem uprawnień zajmuje się sam system na podstawie etykiet przydzielonych do osób i zasobów. Na przykład jeśli osoba ma przydzieloną etykietę „Secret", to może ona przeglądać zasoby oznaczone jako „Secret", a także zasoby na poziomach niższych. Nie może już za to przeglądać zasobów oznaczonych jako „Top Secret". Sam system zapewnia dostęp do zasobów na podstawie ustalonej w trakcie jego tworzenia polityki bezpieczeństwa.

Przypadek **RBAC** bazuje na przydzieleniu dla każdego użytkownika jego roli w organizacji. Przydzielaniem ról i uprawnień do ról zajmuje się administrator. Następnie każdej roli są przydzielane dostępne dla niej zasoby.

Amerykańskie ministerstwo obrony (DoD) zdefiniowało w dokumencie Trusted Computer System Evaluation Criteria [4] (w skrócie TCSEC) następujące 4 poziomy dostępu:
— A – ochrona zweryfikowana
— B – ochrona obowiązkowa
— C – ochrona uznaniowa
— D – ochrona minimalna

Ochrona minimalna dotyczy systemów, które zostały ocenione, ale nie spełniły żadnych kryteriów. Kolejne poziomy oznaczają lepszą ochronę. W wypadku poziomu C możliwa jest tylko ochrona DAC. Dla poziomu B mamy także politykę ochrony, czyli dodatkowo MAC. W wypadku A procedury zostały formalnie zweryfikowane (np. przez podanie dowodu poprawności programu).

W naszym projekcie postanowiliśmy użyć połączenia modeli RBAC i MAC. Z modelu RBAC weźmiemy pojęcie roli. Każdy użytkownik będzie przypisany do określonej roli, której przysługują określone uprawnienia. Jednakże nie będą to uprawnienia statyczne, ale dynamiczne, co oznacza, że w pewnych warunkach pewne uprawnienia będą nieważne lub ograniczone. W tym celu zastosowane zostaną elementy modelu MAC. Inaczej mówiąc model RBAC będzie miał za zadanie zapewnić pewne statyczne uprawnienia, które następnie będą modyfikowane dynamiczne przez odpowiednie zdarzenia zachodzące w systemie i poza nim. Oznacza to, że nasz system uprawnień będzie miał strukturę dwuwarstwową. Pierwszą warstwę będą stanowić przypisane przez administratora systemu role i ich uprawnienia. Nazwijmy takie uprawnienia **możliwościami** roli. To że rola ma możliwość nie oznacza automatycznie, że ma ona także **uprawnienie**. Zdecyduje o tym system na podstawie aktualnych zdarzeń.

Model RBAC został wstępnie opisany w artykule Ferraiolo i Kuhna [6], a dalsze szegóły można znaleźć w artykule Sandhu, Ferraiolo i Kuhna [7]. Amerykański Narodowy Instytut Standardów i Technologii (National Institute of Standards and Technology) opracował model [5], którego celem jest standaryzacja systemów bazujących na rolach. Role stanowią element pośredni, ułatwiający zarządzanie. Do jednej roli może być przyporządkowana więcej niż jedna osoba. Ponadto do jednej roli mogą być przyporządkowane różne zasoby.

Standard NIST podzielił systemy RBAC na 4 klasy, przy czym każda następna klasa jest rozszerzeniem poprzedniej:
— płaski RBAC

— hierarchiczny RBAC
— RBAC z więzami
— symetryczny RBAC

Najprostszą odmianą jest płaski RBAC, który zawiera powiązania między rolami a użytkownikami i uprawnieniami. RBAC hierarchiczny dodatkowo zawiera powiązania między rolami, takie jak na przykład dziedziczenie. W RBAC z więzami dodatkowo jest zapewniona spójność ról i podział odpowiedzialności, a także ograniczenia na powiązania między użytkownikami i rolami, a także pomiędzy jedną rolą a drugą. W symetrycznym RBAC dodatkowo jest możliwość nakładania ograniczeń na powiązania między rolami a uprawnieniami. Jest także możliwość przeglądania i zaawansowanego zarządzania uprawnieniami.

Nasz system chcemy zrealizować jako system z uprawnieniami bazującymi na regułach. Silniki reguł biznesowych to popularne rozwiązanie w firmach. Służą tam one na przykład to zarządzania przepływem dokumentów, a także zarządzania zasobami ludzkimi. Najpopularniejszym rozwiązaniem jest Drools [8], który jest systemem reguł przeznaczonym dla środowiska Java i rozwijanym przez firmę RedHat. Jednakże w naszym projekcie chcemy użyć innego sprawdzonego rozwiązania, jakim jest język Prolog [9]. Jest to język programowania deklaratywnego w logice, który jest bardzo dobrze dostosowany do rozwiązywania skomplikowanych problemów w których występuje wiele reguł.

4. Opis rozwiązania problemu

Jak opisaliśmy powyżej, aby zapewnić naszemu rozwiązaniu maksymalną elastyczność zastosowaliśmy reguły opisane jako reguły Prologowe. Dzięki temu możliwe stało się dowolne ustalenie zależności między uprawnieniami. Zastosowaliśmy model oparty o role, jednakże nasz system pozwala także przypisywać uprawnienia bezpośrednio konkretnym użytkownikom. Co więcej można zupełnie dowolnie kształtować zależności między rolami. Na przykład osobie posiadającej rolę administratora można automatycznie przypisać rolę wikipedysty itp.

Omówiliśmy powyżej sposób organizacji zarządzania uprawnieniami. Jednakże uprawnienia służą przecież do zarządzania określonymi zasobami. Optymalnym rozwiązaniem jest zdefiniowanie ogólnej postaci zasobu, i umożliwieniu zarządzania uprawnieniami do niej. Dzięki postawieniu problemu na takim poziomie ogólności byliśmy w stanie oddzielić proces definiowania i zarządzania uprawnieniami, od procesu tworzenia zasobów. Dzięki temu nasz system uprawnień można dopasować praktycznie do dowolnych zasobów, które jesteśmy w stanie przedstawić w okreslonej postaci ogólnej.

Aby omówić tą postać ogólną należy zauważyć, że każdy zasób, aby był dostępny dla użytkownika za pomocą strony internetowej, musi mieć przypisany mu określony widok, za pomocą którego użytkownik wchodzi w interakcję z tym zasobem. Podstawowymi operacjami, jakie można wykonać na zasobie są operacje czytania zasobu i zapisywania zasobu. Wszystkie inne operacje można rozumieć jako pewne wersje tych dwóch operacji. Jako przy-

kład podamy chociażby dostęp do kolekcji zasobów. Dodawanie elementu do kolekcji to po prostu pisanie do kolekcji. Przeglądanie listy zasobów w kolekcji to odczyt z kolekcji. Tworząc odpowiednią ilość zasobów i bazując na tych dwóch operacjach jesteśmy w stanie opisać praktycznie dowolny zasób.

W związku z powyższym każdą interakcje użytkownika z zasobem możemy opisać za pomocą czterech parametrów zasobu i dwóch parametrów użytkownika:

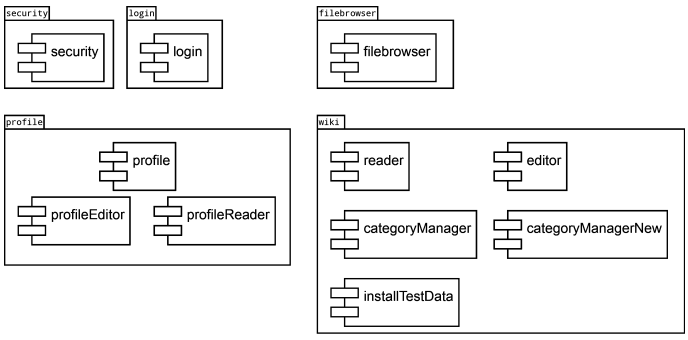
— parametry zasobu:
  — unikalny identyfikator zasobu
  — typ zasobu
  — unikalny identyfikator widoku
  — typ widoku
— parametry użytkownika
  — nazwa użytkownika
  — hasło

Używając tych danych możemy już zaproponować pewien system kontroli dostępu. Jednakże system taki byłby niepełny, gdyż nie możliwe byłoby w nim zdefiniowanie zależności od innych parametrów systemu w aktualnym jego stanie. Dlatego nasz system umożliwia dostęp do dowolnego beana z poziomu Prologu. Dzięki takiemu rozwiązaniu możemy na bierząco podczas weryfikacji uprawnień odpytywać dowolne elementy systemu o ich stan.

5. Opis implementacji

Implementacja rozwiązania problemu została wykonana na platformie JavaEE w technologii JSF i PrimeFaces. Początkowo planowaliśmy użyć bazy danych, jednakże w toku prac okazało się, że takie rozwiązanie jest zbędne i wprowadziłoby tylko niepotrzebne komplikacje. Co więcej nasz system powinien być dostępny dla dowolnych rozwiązań technologicznych, także takich, które nie zawierają bazy danych.

Każdy moduł jest niezależnym komponentem wykonanym przy użyciu technologii JSF Compound Components. Lista komponentów jest przedstawiona na rysunku 1.



Rysunek 1. Diagram komponentów

Każdy komponent jest zestawem widoków, w których można przeglądać określone typy zasobów. Poniżej prezentujemy listę stworzonych komponentów:
— filebrowser – przeglądanie plików na serwerze
— login – logowanie
— profile – przeglądanie profili użytkowników
— security – definiowanie uprawnień
— wiki – funkcjonalnośc wikipedii
Poniżej prezentujemy przykładowe zaimplementowane przez nas reguły dostępowe:

```
isPasswordFor('test','test').
isPasswordFor('no','no').
isPasswordFor('read','read').
isPasswordFor('write','write').
isPasswordFor('admin','admin').
isPasswordFor('zenek','zenek').
isPasswordFor('franek','franek').

hasRole('admin','administrators').
hasRole('zenek','wikipedists').
hasRole('franek','fileadmins').

hasRole(Username,'wikipedists') :- hasRole(Username, 'administrators').
hasRole(Username,'fileadmins') :- hasRole(Username, 'administrators').

canAccessAtIfLogged(user('read','read'),_, readAccess).
canAccessAtIfLogged(user('write','write'),_, writeAccess).

roleCanAccessAtIfLogged('administrators',_, writeAccess).
```

6. Dyskusja rozwiązania i podsumowanie

Nasze rozwiązanie postawionego problemu uznajemy za udane. Jednakże gdybyśmy następnym razem wykonywali podobny projekt, to nie zastosowalibyśmy Javy, ani tym bardziej technologii JSF. Problemem technologii JSF jest jej stanowość. Przystępując do wykonania projektu i wybierając JavęEE i JSF kierowaliśmy się faktem, że wszyscy członkowie zespołu znają język Java i relatywnie łatwe będzie dla nich wdrożenie się w technologię JSF. Jednakża technologia ta nie spełniła pokładanych w niej oczekiwań. Jej głównym problemem jest fakt, że każda odsłona strony powoduje wczytanie zapamiętanego na serwerze drzewa komponentów. Powoduje to, że komponenty te są statyczne i trudno było zmusić tą technologię do dynamicznego podmieniania komponentów pomiędzy wyświetleniami strony. Udało się to w końcu zrealizować, ale dość wysokim kosztem. Dlatego w następnych podobnych projektach nie wykorzystalibyśmy już Javy, ale bardziej dynamiczne środowiska np. Smalltalk i Seaside, Ruby on Rails lub Pythona i Django.

7. Udział w projekcie

Pracami podzieliliśmy się w następujący sposób:
— Mateusz Grotek – projekt systemu, zarządzanie uprawnieniami, Prolog
— Mateusz Jakubczak – komponent profili użytkownika
— Rafał Jurkiewicz – szablon strony i warstwa wizualna
— Łukasz Kotyński – komponent wikipedii
— Paweł Tarasiuk – komponent obsługi plików

Literatura

[1] Access Control Cheat Sheet `https://www.owasp.org/index.php/Access_Control_Cheat_Sheet`
[2] Computer access control `http://en.wikipedia.org/wiki/Computer_access_control`
[3] Access Control Systems w *Fundamentals of Information Systems Security* `http://en.wikibooks.org/wiki/Fundamentals_of_Information_Systems_Security/Access_Control_Systems`
[4] Trusted Computer System Evaluation Criteria `http://en.wikipedia.org/wiki/Trusted_Computer_System_Evaluation_Criteria`
[5] Role Based Access Control (RBAC) and Role Based Security `http://csrc.nist.gov/groups/SNS/rbac/`
[6] Ferraiolo, D. F., Kuhn, D.R., 1992. Role-Based Access Control. 15th National Computer Security Conference
[7] Sandhu, R., Ferraiolo D., Kuhn R. 2000. The NIST Model for Role Based Access Control: Toward a Unified Standard. Proceedings, 5th ACM Workshop on Role Based Access Control, Lipiec 26–27, 2000, Berlin, pp. 47–63
[8] Drools – The Business Logic integration Platform `https://www.jboss.org/drools/`
[9] Kluźniak F., Szpakowicz S. 1985. Prolog For Programmers.