Paolina Centonze IBM T. J. Watson, Hawthorne, NY
Gleb Naumovich Polytechnic University, Brooklyn, NY
Stephen J. Fink IBM T. J. Watson, Hawthorne, NY
Marco Pistoia IBM T. J. Watson, Hawthorne, NY

Role-Based access control consistency validation

International Symposium on Software Testing and Analysis Proceedings of the 2006 international symposium on Software testing and analysis, Portland, Maine,USA
SESSION: Session 4: static analysis, Pages: 121 - 132
Year of Publication: 2006

Seminararbeit von
Oliver GROF
e0227374@student.tuwien.ac.at

Oliver GROF
e0227374@student.tuwien.ac.at

Inhaltsverzeichnis

Einführung

Moderne Enterprise-Systeme, wie zum Beispiel Java EE und Microsoft .NET Common Language Runtime(CLR) unterstützten die rollenbasierte Zugriffskontrolle, den so genannten RBAC(Role-Based Access Control). Der Artikel präsentiert einen theoretischen Ansatz um die operationsbasierte und die datenbasierte Strategie zu verbinden. Die zwei Autoren Gleb Naumovich und Paulina Centonze haben schon 2004 in [2],einer früheren Arbeit zu dem Thema betont, dass die Absicht einer Sicherheitsstrategie (*security policy*) oft ist, privilegierte Daten im Gegensatz zu Operationen zu schützen. Um das RBAC auf die „location consistency" zu überprüfen bzw. zu validieren wird eine Technik zur statischen Analyse vorgestellt. Ein weiterer neuer Beitrag des Artikels ist das Design und die Implementierung eines Analyse-Tool, das den Namen SAVES (Static Analysis for Validation of Enterprise Security) trägt. Das Paper behandelt die Herausforderungen der Analyse von Java EE Applikationen und präsentiert auch experimentelle Studien. Ein erfreuliches Ergebnis ist, das viele Mängel bezüglich methodenbasiertes RBAC aufgedeckt und keine „false positives" geliefert werden. Da die Aufgabe darin bestand, einen weiterführenden Artikel aus dem Themenbereich „statische Analyse in Java" zu suchen, behandle ich in meiner Seminararbeit die Sicherheitsanalyse in Java EE und die Implementierung des Tools „SAVES" detaillierter, als die theoretisch-mathematischen Gedankenfolgen über dem RBAC.

Motivationsbeispiele

Die Java EE Spezifikation erlaubt es, Zugriffseinschränkungen für bestimmte Methoden der Enterprise JavaBeans (EJB) Komponenten zu setzen. Abbildung 1 zeigt einen Teil eines Enterprise Beans. Die Komponente StudentBean enthält die Methoden *setGrade* und *setProfile*. Der Systemadministrator könnte jetzt *setGrade* mit der Rolle Professor beschränken, und *setProfile* mit der Rolle Student. Da der Quellkode oft nicht bekannt ist, wäre die Konfiguration möglich, und somit könnten die Benutzer mit der Rolle Student auch mit setProfile auf Daten zugreifen, auf die sie nicht sollten.

```
// package and import declarations here...
public class StudentBean implements SessionBean {
    private String name, address;
    private Map grades = new HashMap();
    public void setGrade(String c, Character g) {
        grades.put(c, g);
    }
    public void setProfile(String n, String a,
            Map m) {
        this.name = n;
        this.address = a;
        this.grades = m;
    }
    // other code here...
}
```

Abbildung 1: Fragment des `StudentBean`, Quelle: [1]

Das Problem ist also, dass aufgrund von fehlerhaften RBAC – Strategien die Konsistenzeigenschaft verletzt wird. Bei Webapplikationen, die meist eine weitaus komplexere Struktur aufweisen, sind die Inkonsistenzen oft schwerer zu finden. Abbildung 2 zeigt zum Beispiel eine Strategie, bei der mit der Rolle *q* auf das sicherheitssensitive Feld *f* (und die dazugehörigen Daten), zwar nicht direkt über die Methode m_3, aber z.B. indirekt über die Methode m_4 zugegriffen werden kann.

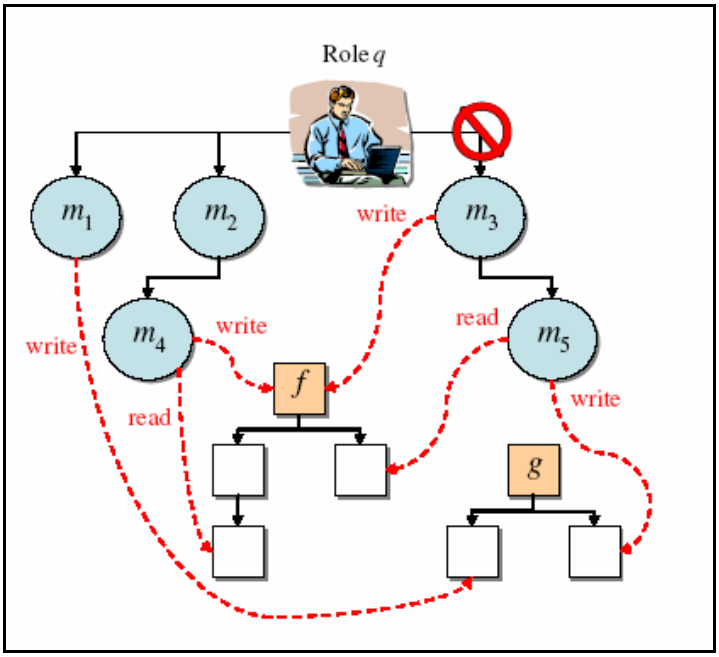

Abbildung 2: RBAC Policy Scenario, Quelle: [1]

Im Extremfall müssen eventuell alle Methoden untersucht werden, daher ist ein automatisierter Ansatz sinnvoll.

Terminologie

Nachfolgend wird anhand einiger Begriffserklärungen die Terminologie des Artikels nahe gebracht.

Rolle (role): a set of access rights that can be assigned to users and groups of a computer system

***rollenbasierte Zugriffskontrolle* (*RBAC*):** ist in Mehrbenutzersystemen oder Rechnernetzen ein Verfahren zur Zugriffssteuerung und -kontrolle auf Dateien oder Dienste, bei dem Benutzern des Computers oder Netzwerks Rollen zugeordnet werden.

***location consistency*:** eine Eigenschaft, die anzeigt, ob eine gegebene operationsbasierte RBAC – Strategie zu irgendeiner datenbasierter RBAC – Strategie äquivalent ist

***SAVES**:* Static Analysis for Validation of Enterprise Security. SAVES analysiert den Java EE bytecode, prüft die assoziierte RBAC – Strategie auf "location consistency", meldet potentielle Sicherheitslücken, wo die "location consistency" nicht gewährleistet ist

Das RBAC Consistency Model

In dem Paper beschreiben die Autoren ein präzises, formales Modell zur Überprüfung der Konsistenz. Formuliert werden die verschiedenen Wege, in denen das Programm auf Daten zugreifen kann und es wir ein Netz vorgestellt, das auf Operationen basiert, die von Programmmethoden ausgeführt werden. Dieses erlaubt die Untergliederung der Methoden in so genannte Equivalenzklassen. Wichtig ist noch, dass der Begriff des *location-based consistency* für die methodenbasierte RBAC - Strategie charakterisiert wird. Der Begriff der *location consistency* beschreibt im Prinzip die Eigenschaft, dass eine methodenbasierte RBAC – Strategie ein kompatibles Datenschutzschema verkörpert.
In dieser Seminararbeit werden wir auf folgende Definition zurückgreifen:

*If **μ** is a method-based RBAC policy for **p**, the location-based RBAC policy* induced *with **μ** is the function **Λμ**.*

Rollenbasierte Zugriffskontrolle in Java EE

Die Java EE Spezifikation schreibt vor, dass bei einem Zugriff von einer Komponente „B" auf eine eingeschränkte Ressource in einer Komponente A – wie zum Beispiel eine Methode in einem Enterprise Bean- ein „authorization check", also eine Berechtigungsprüfung stattfindet. Die User und die Gruppen sind auf der System-Ebene definiert, die Rollen jedoch sind Applikationsspezifisch; jede Java EE Applikation definiert ihre eigene Sicherheitsrollen (*security roles*).
Java EE bietet eine Zugriffskontrolle für EJB Komponenten. Die Methoden eines enterprise beans sind in den so genannten EJB-Klassen implementiert. Damit ein Klient (wie zum Beispiel ein anderes enterprise bean, ein servlet oder eine standalone application) auf eine EJB Methode zugreifen kann, muss diese Methode in einem spezifischen EJB Interface deklariert werden. Es gibt vier Sorten EJB Interfaces: Remote, RemoteHome, Local und LocalHome. Die Methoden, die in Remote und RemoteHome implementiert sind können von Clientprogrammen in anderen Containern, Methoden die in Local und LocalHome implementiert sind können von Clientprogrammen im selben Container aufgerufen werden.
Das Autorisierungsmodell von Java EE unterscheidet sich in einem wichtigen Punkt von dem formalen Modell im letzten Abschnitt. Java EE unterwirft nur die

intercomponent calls der Berechtigungsprüfung, die Zugriffskontrolle wird also nicht an *helper* Methoden oder *calls* innerhalb derselben EJB Komponente angewandt. Diese Ausnahme macht die Argumentation der RBAC - Strategie komplizierter und kann zu Sicherheitslücken führen, wenn man diese nicht ordentlich behandelt.

Deklarative Sicherheit

Dies ist ein Konzept, das sowohl von Java EE, als auch von CLR gefördert wird. Es ist hierbei nicht nötig, den Autorisierungscode in die Applikation einzubetten. In Java EE gibt es zu diesem Zweck *deployment descriptor*, der mit der Sprache XML definiert wird. Eine Beschreibung der Rolle `Professor` sehen Sie in Abbildung 1.

```
<assembly-descriptor>
   <security-role>
      <role-name>Professor</role-name>
   </security-role>
   <method-permission>
      <role-name>Professor</role-name>
         <method>
            <ejb-name>StudentBean</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>setGrade</method-name>
         </method>
   </method-permission>
</assembly-descriptor>
```

Abbildung 3: Ein sicherheitsbezogener Teil des deployment descriptors, Quelle: [1]

In der Defaultkonfiguration können alle Rollen auf alle Methoden zugreifen, die nicht explizit im *deployment descriptor* der Applikation eingeschränkt sind. Solche Methoden werden *unchecked* genannt.

Statische Analyse von Java EE Applikationen

Die statische Analyse erfolg also durch das Tool SAVES. Dieses wurde von den Autoren in Java SE implementiert und kontrolliert die location consistency für die methodenbasierte Java EE RBAC Strategie.
SAVES bekommt als Input das Programm ***p***, das eine oder mehrere Java-Applikationen beinhaltet. All diese Applikationen sind eine Kollektion von EAR(Enterprise Archive) Files, oder eben ein JAR(Java Archive) und WAR(Web Archive) Files. SAVES bekommt die methodenbasierte RBAC Strategie ***μ*** für ***p*** von der deployment description, die in den Archivfiles eingebettet ist. Die Schritte der SAVES-Analyse können im Anhang nachgelesen werden.

Empirische Evaluation von SAVES

Die Evaluation von SAVES wurde mit verschiedenen Applikationen durchgeführt, wobei es einen vertretbarem Zeitaufwand und Speicherplatzbedarf gab, und bis zu 15 Inkonsistenzen gefunden wurden.

Name	Size (KB)		Methods			Time (sec)	Mem. (MB)	Roles	Inco.
	EAR	EJB	Total	App.	Bus.				
Trade3	1,076	136	5,438	677	48	152.11	218	3	15
ITSOBank	302	205	2,323	191	22	64.97	255	2	2
DukesBank	128	34	1,579	188	21	118.67	160	2	2
Bookstore	359	33	12,178	256	13	60.00	330	3	3
EnrollerApp	15	12	2184	55	12	65.71	252	3	9
SavingsAcc	10	7	2154	19	5	64.50	250	3	4
PetStore	1,282	133	6,411	339	36	255.22	300	2	0
CartApp	7	5	27	9	3	20.66	133	2	2

Tabelle 1: Experimentelle Ergebnisse von SAVES, Quelle: [1]

Zusammenfassung

Ein Anliegen der Autoren war es, auf die Wichtigkeit der Zugriffskontrolle auf sicherheitssensitiven Feldern (Daten) aufmerksam zu machen. Bevor auf die rollenbasierte Zugriffskontrolle in Java EE eingegangen wurde, erfolgte also eine Gegenüberstellung von der operationsbasierten und der datenbasierten RBAC-Strategie. Schließlich wurde das SAVES, das entwickelte Tool zur statische Analyse, vorgestellt und evaluiert.

Literaturverzeichnis

[1] Paolina Centonze, Gleb Naumovich, Stephen J. Fink, Marco Pistoia. Role-Based access control consistency validation. International Symposium on Software Testing and Analysis Proceedings of the 2006 international symposium on Software testing and analysis, Portland, Maine,USA SESSION: Session 4: static analysis, Pages: 121 – 132, 2006

[2] Gleb Naumovich and Paolina Centonze. Static Analysis of Role-Based Access Control in J2EE Applications. SIGSOFT Software Engineering Notes, 29(5):1–10, September 2004.

Anhang A: Schritte der der SAVES Analyse

Die Schritte wurden anhand der Beschreibung in [1] in Punkte gesammelt. Sie wurden bewusst Englisch gehalten um keinen Informationsverlust durch die Übersetzung zu riskieren.

1., Perform a flow-insensitive pointer analysis and build call graph G = (N,E) SAVES consults the deployment descriptors to get the relevant Information (such as EJB interface methods, set of roles that are allowed access to that method)

2., SAVES uses the pointer analysis abstractions to partition the memory locations into abstract locations

3., With the call graph and pointer analysis in hand, SAVES performs a contextinsensitive interprocedural mod-ref analysis to determine the access tuple solution for each method.

4., For each $n \in N$ representing the invocation of a method $m_n \in M$, SAVES determines the EJB fields that are directly read and directly written by m_n, and uses the points-to graph to determine the EJB fields that are directly partially read and directly partially written by m_n.

5., SAVES computes the special logical functions defined in RBAC.

6., Finally, SAVES detects potential location inconsistencies for μ

→ SAVES outputs the set of potential inconsistencies OR

→ SAVES outputs the overapproximation of $\Lambda\mu$.

(see "RBAC Consistency Model" for the definition of $\Lambda\mu$)