Verwaltung von Java-2-Zugriffspolitiken

Rainer Falk

Lehrstuhl für Datenverarbeitung
TU München
falk@ei.tum.de

Zusammenfassung. In Java 2 können für mobilen und lokalen Code feingranular Zugriffsrechte vergeben werden. Die Zugriffspolitik legt abhängig von der Code-Herkunft fest, auf welche Ressourcen zugegriffen werden darf.
In diesem Beitrag wird eine Methode vorgestellt, um auch komplexe Java-2-Zugriffspolitiken für eine große Anzahl von Systemen effizient verwalten zu können. Die in einem LDAP-Verzeichnisdienst abgelegten Zugriffspolitiken werden strukturiert, indem Politiken für Teilbereiche kombiniert werden. Die Administration der Zugriffspolitiken kann dezentral erfolgen, indem die Verantwortung für eine Teilpolitik delegiert wird. Beim Einbinden einer Teilpolitik kann eingeschränkt werden, für welche Code-Herkunft Zugriffsrechte vergeben werden können.

1 Einleitung

Java 2 bietet die Möglichkeit, feingranular Zugriffsrechte abhängig von der Herkunft einer Klasse (*Code Source*) zu spezifizieren. Die Herkunft ist durch die URL, über die eine Klasse geladen wurde, und durch die Signierer der Klasse bestimmt. In der Standardimplementierung wird eine einfache Textdatei zur Spezifikation der gewährten Zugriffsrechte verwendet.

In einem Netzwerk mit einer großen Anzahl von Rechnern, Benutzern und Java-Anwendungen (sowohl Applets als auch Applikationen) müssen viele Politikbeschreibungen aktuell gehalten werden. Der dazu notwendige Verwaltungsaufwand kann unakzeptabel hoch sein. Hinzu kommt, daß in mehreren Politiken oft identische Teilpolitiken vorkommen. Wenn Änderungen erforderlich sind, z. B. weil eine aktualisierte Version einer Anwendung andere Zugriffsrechte benötigt oder wenn neue Anwendungen verwendet werden, müssen viele Politiken aktualisiert werden.

Die Verwaltung der Java-2-Zugriffspolitiken kann vereinfacht werden, indem die einzelnen Politiken nicht als eine Einheit spezifiziert werden. Stattdessen werden sie aus Teilpolitiken zusammengesetzt. Diese können von mehreren Administratoren dezentral verwaltet werden. Die Politiken werden in einem LDAP-Verzeichnisdienst (*Lightweight Directory Access Protocol*) abgelegt. Beim Einbinden einer Teilpolitik kann eingeschränkt werden, für welche Code-Herkunft Ressourcenzugriffe autorisiert werden können.

Der Beitrag ist wie folgt aufgebaut: Abschnitt 2 beschreibt das Sandbox-Sicherheitsmodell und das Sicherheitsmodell von Java 2. In Abschnitt 3 werden

rollenbasierte Prinzipien zur Rechteverwaltung kurz vorgestellt. Auf diesen basiert das Einbinden von Teilpolitiken, das in Abschnitt 4 beschrieben wird. Abschnitt 5 stellt das Schema für die Ablage von Java-2-Zugriffspolitiken in einem LDAP-Server und den implementierten Prototypen vor. Abschnitt 6 beschreibt, wie eine organisationsweit definierte Zugriffspolitik durchgesetzt werden kann. Abschnitt 7 endet mit einer Zusammenfassung.

2 Java-Sicherheitsmodell

2.1 JDK 1.0.2 und JDK 1.1

Das ursprünglich bei Java verwendete Sicherheitsmodell ist als Sandbox-Modell bekannt [4]. Dort wird Code als vertrauenswürdig oder als nicht vertrauenswürdig eingestuft. Vertrauenswürdiger Code darf uneingeschränkt auf Systemressourcen (z. B. Dateisystem, Netzwerkverbindungen) zugreifen, während nichtvertrauenswürdiger Code nur auf wenige Ressourcen, die von der Sandbox bereitgestellt werden, zugreifen kann. Beim Sandbox-Modell sind die Unterscheidung in vertrauenswürdigen und nicht vertrauenswürdigen Code und die damit verbundenen Zugriffsrechte fest vorgegeben.

2.2 Java 2

Bei Java 2 gibt es keine festgelegte Unterscheidung zwischen vertrauenswürdigem und nicht vertrauenswürdigem Code. Stattdessen kann abhängig von der Code-Herkunft flexibel festgelegt werden, auf welche Ressourcen zugegriffen werden darf [4]. Die Code-Herkunft ist durch das verwendete Protokoll (z. B. HTTP), den Rechner, die Portnummer, den Dateipfad, über den eine Klasse geladen wurde, und den Signieren der Klasse bestimmt. Diese Sicherheitsarchitektur bietet folgende Möglichkeiten:

- *Feingranulare Zugriffskontrolle für alle Java-Programme.* Damit kann erreicht werden, daß lokaler und vom Netz geladener Code nur über die notwendigen Zugriffsrechte verfügt.
- *Konfigurierbare Sicherheitspolitik.* Auf welche Ressourcen eine Anwendung zugreifen kann, ist nicht fest vorgegeben, sondern kann von einem Benutzer oder Systemverwalter definiert werden.
- *Erweiterbare Zugriffskontrollstruktur.* Eine Anwendung kann eigene Zugriffsrechte definieren, die zusammen mit den von Java 2 definierten Zugriffsrechten einheitlich verwaltet werden können.

Durchsetzung zur Laufzeit. Eine geladene Klasse wird einem Schutzbereich (*Protection Domain*) zugeordnet. Alle Klassen, die dem gleichen Schutzbereich zugeordnet sind, verfügen über die gleichen Zugriffsrechte (vgl. Abb. 1).

Um die Zugriffsrechte einer Klasse zu ermitteln, wird die Methode `getPermissions(codesource)` der abstrakten Klasse `java.security.Policy` aufgerufen. Diese bestimmt die Menge aller Zugriffsrechte, die Code mit der Herkunft

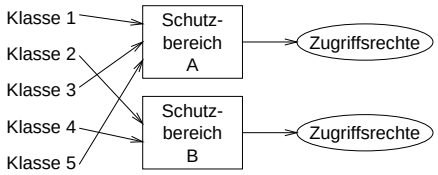

Abb. 1. Klassen, Schutzbereiche und Zugriffsrechte

`codesource` besitzt. Die in Java 2 vorhandene konkrete Implementierung der Klasse `Policy` liest die Politik aus einer Textdatei. Diese enthält einen *Keystore*-Eintrag, der angibt, wo die öffentlichen Schlüssel der im folgenden verwendeten Signierer zu finden sind. Es folgen beliebig viele Politikeinträge (*Grant Entry*). Ein Politikeintrag gewährt für die angegebene Code-Herkunft eine Menge von Zugriffsrechten.

```
grant codeBase "http://intraserv.in.de/apps/bib/-", signedBy "dbadmin" {
  java.net.SocketPermission "db.in.de:3000", "connect";
  java.lang.RuntimePermission "queuePrintJob";
};
```

Abb. 2. Politikeintrag (Beispiel)

Abbildung 2 zeigt ein Beispiel für einen Politikeintrag. Es werden Zugriffsrechte für Klassen gewährt, die von `http://intraserv.in.de/apps/bib/` oder einem Unterverzeichnis davon geladen wurden und die von `dbadmin` signiert sind. Solche Klassen dürfen eine Verbindung zu dem Rechner `db.in.de` auf der Portnummer 3000 aufbauen. Weiterhin dürfen diese Klassen einen Druckauftrag absetzen.

Die Angaben für `codeBase` oder `signedBy` können entfallen. Die angegebenen Zugriffsrechte werden dann allen Klassen unabhängig von der URL, von der sie geladen wurden, bzw. unabhängig von den Signierern gewährt.

Um für eine Klasse die gewährten Zugriffsrechte zu bestimmen, werden die in einer Politikdatei enthaltenen Politikeinträge der Reihe nach ausgewertet. Wenn die angegebenen Bedingungen für die Code-Herkunft erfüllt sind, werden die enthaltenen Zugriffsrechte gewährt. Die Menge der gewährten Zugriffsrechte ist die Vereinigungsmenge der von einzelnen Politikeinträgen gewährten Zugriffsrechte.

Die Konfigurationsdatei `${java.home}/lib/security/java.security` legt u. a. fest, welche Klasse zur Bestimmung der gewährten Zugriffsrechte verwendet wird und welche Politik-Dateien eingelesen werden. In der Standardkonfiguration werden eine systemweite und eine benutzerspezifische Politik ausgewertet.

3 Rollenbasierte Rechteverwaltung

Die Verwaltung von Zugriffsrechten für eine große Benutzerpopulation stellt eine große Herausforderung dar. Im kommerziellen Umfeld hat sich insbesondere eine rollenbasierte Rechteverwaltung als vorteilhaft erwiesen [2].

Bei einer rollenbasierten Rechteverwaltung werden Zugriffsrechte nicht individuellen Benutzern zugeordnet, sondern Rollen [10]. Eine Rolle entspricht einer Menge von Zugriffsrechten. Einer Rolle können einzelne Zugriffsrechte zugeordnet werden, oder sie kann die Zugriffsrechte einer anderen Rolle übernehmen. Rollen werden so definiert, daß sie den Aufgaben der Mitarbeiter entsprechen. Ein Mitarbeiter erhält die für seinen Aufgabenbereich erforderlichen Zugriffsrechte, indem er den entsprechenden Rollen zugeordnet wird.

Abbildung 3 illustriert den Zusammenhang zwischen Zugriffsrechten, Rollen, Rollenhierarchie und Benutzern. Christian ist der Rolle „Projektleiter Projekt A“

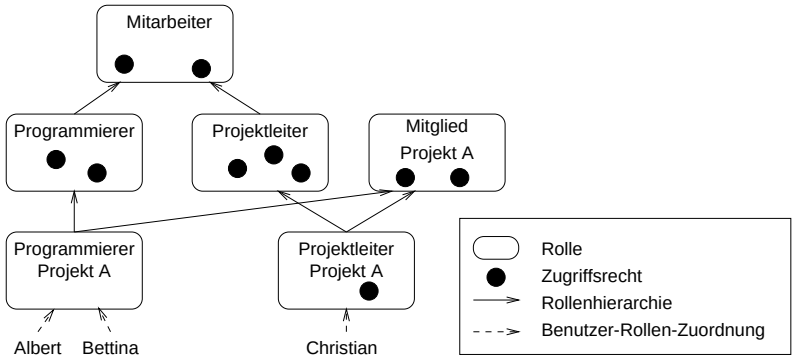

Abb. 3. Rollenbasierte Rechteverwaltung

zugeordnet. Dadurch erhält er die Zugriffsrechte der Rolle „Projektleiter Projekt A“ und durch die Rollenhierarchie die Zugriffsrechte der Rollen „Mitglied Projekt A“, „Projektleiter“ und „Mitarbeiter“.

4 Effiziente Verwaltung

Die Grundidee einer rollenbasierten Rechteverwaltung liegt darin, Zugriffsrechte so zu gruppieren, daß die Gruppen eine bestimmte höhere Bedeutung haben. Eine Gruppe von Zugriffsrechten ist nicht einfach eine beliebige Menge von Zugriffsrechten, sondern sie enthält diejenigen Zugriffsrechte, die zur Durchführung einer Aufgabe erforderlich sind.

Die Zugriffskontrolle von Java 2 unterscheidet sich jedoch in folgenden Punkten von einer Verwaltung von Benutzerrechten:

– Die Zugriffsrechte werden letztendlich nicht Benutzern zugeordnet, sondern Programmcode. Es geht bei der Zugriffskontrolle nicht darum, die Zugriffsmöglichkeiten für Benutzer einzuschränken, sondern der Benutzer (oder ein Systemverwalter) möchte die Zugriffsmöglichkeiten für mobilen Code unterschiedlicher Herkunft festlegen.
– Es gibt keine zentrale Instanz, die die Zugriffsrechte festlegt, sondern jeder Benutzer kann seine eigene Zugriffspolitik definieren.

In einem Intranet mit vielen Benutzern und Rechnern weist das Verwaltungsmodell der Standardimplementierung folgende Probleme auf:

– Es muß für jeden Benutzer und jeden Rechner eine Politik definiert werden. Diese sollte nur die jeweils erforderlichen Zugriffsrechte gewähren. Der damit verbundene Verwaltungsaufwand ist insbesondere in einer dynamischen Umgebung hoch, da bei Änderungen – z. B. durch die Installation neuer Programme oder geänderter Programmversionen – viele Politiken aktualisiert werden müssen.
– Ein Anwender, der kein Sicherheitsexperte ist, ist nicht in der Lage, die Konsequenzen einer Zugriffspolitik zu beurteilen. Von der spezifizierten Sicherheitspolitik ist nicht nur die Sicherheit des jeweiligen Arbeitsplatzrechners betroffen, sondern das gesamte Intranet. Eine vorhandene Firewall kann nicht vor Angriffen schützen, die von mobilem Code ausgehen, der auf einem Rechner im geschützten Netz ausgeführt wird.
– Bei einer komplexen Zugriffspolitik ist nur schwer nachvollziehbar, welche Zugriffsrechte tatsächlich gewährt werden.

Im folgenden wird ein Konzept vorgestellt, um Java-2-Zugriffspolitiken auch in größeren Netzen effizient spezifizieren zu können. Dieses bietet folgende Vorteile:

– Es besteht die Möglichkeit, von zentraler Stelle aus alle im Gesamtnetz benötigten Zugriffspolitiken zu definieren.
– Eine Politik kann durch das Einbinden von Teilpolitiken modularisiert werden. Eine Teilpolitik kann dabei als Baustein in mehreren Politiken verwendet werden. Durch die bessere Strukturierung ist die spezifizierte Politik leichter verständlich. Die Wahrscheinlichkeit von Fehlern wird verringert, und ein Überprüfen der Zugriffspolitik wird erleichtert.
– Die Administration von Teilbereichen kann auf mehrere Administratoren verteilt werden. Die Code-Herkunft, für die in einer Teilpolitik Zugriffsrechte gewährt werden können, kann festgelegt werden. Dadurch werden die Auswirkungen eines Fehlers beschränkt.

4.1 Erweiterung von Politikeinträgen

Bei dem in diesem Beitrag vorgestellten Konzept werden Politikeinträge erweitert: Ein Politikeintrag kann nicht nur eine Menge von Zugriffsrechten enthalten, sondern zusätzlich auch einen Verweis auf weitere Politikeinträge.

Um die Zugriffsrechte für eine Code-Herkunft zu ermitteln, wird für einen Politikeintrag wie bisher überprüft, ob die angegebenen Kriterien für die Code-Herkunft zutreffen. Im positiven Fall werden die im betrachteten Politikeintrag angegebenen Zugriffsrechte gewährt, und es werden rekursiv die Politikeinträge ausgewertet, auf die verwiesen wird. Andernfalls werden weder die Zugriffsrechte gewährt, noch werden die angegebenen Politikeinträge bearbeitet. Im Gegensatz zur Standardimplementierung werden dabei nicht alle vorhandenen Politikeinträge ausgewertet.

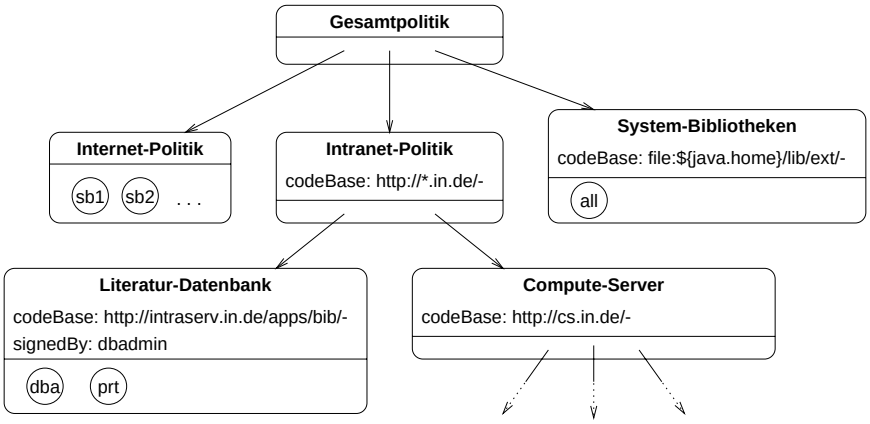

Abb. 4. Beispielpolitik

Beispiel. Das Einbinden von Politikeinträgen soll anhand der in Abb. 4 dargestellten Beispielpolitik erläutert werden. Ein Kasten stellt jeweils einen Politikeintrag dar. Oben steht fettgedruckt der Name des jeweiligen Politikeintrags. Durch `codeBase` und `signedBy` werden die Kriterien angegeben, die die Code-Herkunft erfüllen muß, damit ein Politikeintrag ausgewertet wird. Im unteren Teil, der durch einen Strich abgetrennt ist, werden die gewährten Zugriffsrechte durch Kreise dargestellt. Mit Pfeilen werden Verweise auf andere Politikeinträge symbolisiert.

Die *Gesamtpolitik* wird definiert, indem drei Teilpolitiken eingebunden werden. In *Gesamtpolitik* sind keine Einschränkungen für die Code-Herkunft angegeben. In den eingebundenen Politikeinträgen können also Zugriffsrechte für eine beliebige Code-Herkunft vergeben werden. Der linke Politikeintrag *Internet-Politik* gewährt unabhängig von der Code-Herkunft die Rechte *sb1*, *sb2*... Diese entsprechen den Zugriffsrechten, die Code besitzt, der bei JDK 1.0.2 in der Sandbox ausgeführt wird. Der rechte Politikeintrag *System-Bibliotheken* gewährt System-Bibliotheken das Zugriffsrecht *all*, das uneingeschränkten Zugriff auf alle Ressourcen ermöglicht. Der Politik-Eintrag *Intranet-Politik* gruppiert Politik-

einträge für mehrere Intranet-Anwendungen. Dieser Eintrag gewährt selbst keine Zugriffsrechte. Die eingebundenen Politikeinträge *Literatur-Datenbank* und *Compute-Server* werden allerdings nur ausgewertet, wenn die Zugriffsrechte für eine Klasse bestimmt werden, die über HTTP von einem Rechner der Domäne `in.de` geladen wurde. Der Politikeintrag *Literatur-Datenbank* gewährt der Anwendung für den Zugriff auf eine Literaturdatenbank die erforderlichen Zugriffsrechte. Dazu muß der Code über HTTP vom Intranet-Server `intraserv.in.de` aus dem Pfad `/apps/bib/` (oder einem Unterverzeichnis) geladen werden und er muß von `dbadmin` signiert sein. Diesem werden die Zugriffsrechte `dba` zum Netzzugriff auf die Datenbank und `prt` zum Absetzen von Druckaufträgen gewährt. Entsprechend gewährt der Politikeintrag *Compute-Server* die Zugriffsrechte, die Anwendungen benötigen, die von dem Compute-Server `cs.in.de` geladen werden. Durch die Pfeile wird angedeutet, daß auf weitere, hier nicht dargestellte Politikeinträge verwiesen wird. Diese gewähren die jeweils erforderlichen Zugriffsrechte für unterschiedliche Anwendungen auf dem Compute-Server. In diesen Politikeinträgen können nur Rechte für Code gewährt werden, der über HTTP von `cs.in.de` geladen wurde. Selbst wenn in diesen Politikeinträgen für die Code-Herkunft keine Einschränkungen angegeben sind, werden diese Politikeinträge nur ausgewertet, wenn die Zugriffsrechte für Code bestimmt werden, die von dem Compute-Server geladen wurden.

5 Implementierung

Es wurde ein Prototyp entwickelt, bei dem die Zugriffspolitik in einem LDAP-Server abgelegt wird. Dazu wurde ein Schema definiert, das die Struktur der Information beschreibt. Für die Auswertung der im LDAP-Server abgelegten Politik wurde eine Unterklasse von `java.security.Policy` implementiert.

5.1 Ablage in Verzeichnisdienst

Bei LDAP wird die Information durch Einträge (*Entries*) dargestellt [5]. Ein Eintrag kann über einen eindeutigen Namen, den *Distinguished Name*, angesprochen werden. Ein Eintrag besteht aus mehreren Attributen, denen Werte zugeordnet sind. Durch Klassendefinitionen wird festgelegt, welche Attribute in einem Eintrag vorhanden sein müssen und welche vorhanden sein können.

Um die Zugriffspolitik von Java 2 in einem LDAP-Server ablegen zu können, wurden die in Abb. 5 gezeigten Klassen zur Darstellung von Zugriffsrechten (`javaPermission`), Politikeinträgen (`javaGrantEntry`) und einer Zugriffspolitik (`javaPolicy`) definiert. Das Attribut `cn` (*Common Name*) enthält den Namen des Objekts. Dieser ist Teil des *Distinguished Name*, über den das Objekt angesprochen werden kann. Im Attribut `description` kann ein erläuternder Kommentar abgelegt werden.

Ein Zugriffsrecht wird durch einen Eintrag der Klasse `javaPermission` dargestellt. Die Attribute `javaPermissionClass`, `javaPermissionTarget`, `javaPermissionAction` und `signedBy` spezifizieren das Zugriffsrecht.

javaPolicy	**javaGrantEntry**	**javaPermission**
cn	cn	cn
description	description	description
javaKeystore	codeBase	javaPermissionClass
javaKeystoreType	signedBy	javaPermissionTarget
grantEntry	permission	javaPermissionAction
	grantEntry	signedBy

Abb. 5. LDAP-Klassendefinitionen

Ein Eintrag der Klasse `javaGrantEntry` ist ein Politikeintrag. Die Attribute `codeBase` und `signedBy` spezifizieren die Code-Herkunft, für die ein Eintrag Zugriffsrechte gewährt. Das Attribut `permission` enthält die *Distinguished Names* der gewährten Zugriffsrechte. Über die Einträge des Attributs `grantEntry` kann auf weitere Politikeinträge verwiesen werden.

Eine Zugriffspolitik wird durch einen Eintrag der Klasse `javaPolicy` dargestellt. Durch die Attribute `javaKeystore` und `javaKeystoreType` wird festgelegt, wo und in welchem Format die öffentlichen Schlüssel der Signierer abgelegt sind. Das Attribut `grantEntry` enthält die *Distinguished Names* der zu einer Politik gehörenden Politikeinträge.

5.2 Zugriff auf Politik

Es wurde ein Prototyp einer Unterklasse von `java.security.Policy` entwickelt, die die Politikbeschreibung von einem LDAP-Server lädt. Als Parameter benötigt sie die Adresse und die Portnummer des LDAP-Servers und den *Distinguished Name* des gewünschten Politik-Objekts.

Für den Zugriff auf den LDAP-Server wurde das *Java Naming and Directory Interface 1.2* [13] verwendet. Als LDAP-Server kam *OpenLdap* [8] zum Einsatz.

5.3 Sicherheit des Verwaltungssystems

Die Sicherheit des Verwaltungssystems basiert auf den Sicherheitsmechanismen, die von dem Verzeichnisdienst bereitgestellt werden.

Kommunikationssicherheit. Die Verbindung zum LDAP-Server kann z. B. durch SSL (*Secure Sockets Layer*) [3] gesichert werden. Damit können der Server und der Client authentifiziert werden, und auf dem Übertragungsweg kann ein Mitlauschen oder Modifizieren der übertragenen Daten verhindert werden. Der verwendete LDAP-Server unterstützt jedoch – im Gegensatz zu kommerziell verfügbaren Produkten – kein SSL.

Zugriffskontrolle auf Teilpolitiken. Durch die von einem LDAP-Server angebotenen Zugriffskontrollmechanismen kann eingeschränkt werden, wer auf welche

Teile der im Verzeichnisdienst abgelegten Information in welcher Art zugreifen kann. Damit kann flexibel festgelegt werden, welcher Benutzer eine (Teil-)Politik anlegen, verändern oder auch nur lesen darf.

6 Durchsetzung einer organisationsweiten Sicherheitspolitik

Bei dem Administrationsmodell der Standardimplementierung kann jeder Benutzer selbst eine Java-2-Zugriffspolitik definieren. Ein Anwender, der i. a. kein Sicherheitsexperte ist, kann jedoch die Auswirkungen einer Sicherheitspolitik nicht beurteilen. Deshalb ist es in Firmennetzen sinnvoll, daß die Zugriffspolitik nur von einem Sicherheitsadministrator festgelegt werden kann. Auf den einzelnen Arbeitsplatzrechnern muß sichergestellt werden, daß diese Politik auch verwendet wird.

Bei Java 2 ist in einer systemweiten Konfigurationsdatei angegeben, welche *Policy*-Klasse verwendet werden soll und welche Politikbeschreibungen ausgewertet werden sollen. An dieser Stelle kann festgelegt werden, daß anstatt der Standardimplementierung die in Abschnitt 5 beschriebene Implementierung der Klasse *Policy* verwendet werden soll. Außerdem kann die auszuwertende Politik angegeben werden. Durch Dateizugriffsrechte, soweit sie von dem verwendeten Betriebssystem unterstützt werden, kann sichergestellt werden, daß nur Administratoren die Konfiguration ändern können. Ein Benutzer kann die festgelegte Konfiguration jedoch durch Aufrufoptionen umgehen. Um dies zu verhindern, muß der direkte Aufruf des Java-Laufzeitsystems unterbunden werden. Stattdessen kann ein Programm aufgerufen werden, das das Java-Laufzeitsystem erst nach einer Überprüfung und eventuellen Modifikation der Aufrufparameter startet.

Diese Sicherheitsmaßnahmen greifen nur, wenn die Benutzer auf ihren Arbeitsplatzrechnern keine Programme installieren können und wenn sie die Konfiguration des Java-Laufzeitsystems nicht ändern können. Ein anderer Ansatz kann für Applets verwendet werden, die aus dem Internet über eine Firewall geladen werden. Bei einem solchen Szenario kann ein Client im Intranet keine direkte Verbindung zu einem Rechner im Internet aufbauen, sondern nur zu einem Proxy-Server. Dieser lädt die angeforderten Dateien – HTML-Dateien und Java-Applets – und reicht sie an den Client weiter. Der Proxy-Server kann jedoch statt des angeforderten Applets zuerst ein Prüf-Applet an den Client schicken, das die Konfiguration des Clients überprüft. Neben der Integrität der sicherheitsrelevanten Komponenten der Java-Laufzeitumgebung kann auch überprüft werden, ob tatsächlich die an zentraler Stelle definierte Zugriffspolitik verwendet wird. Nur wenn die Prüfungen erfolgreich verlaufen sind, wird das eigentlich gewünschte Applet an den Client weitergeleitet. Die Überprüfung führt jedoch zu Verzögerungen. Um die Benutzerakzeptanz zu erhöhen, kann es sinnvoll sein, die Überprüfung nicht bei jedem Laden eines Applets durchzuführen, sondern nur in regelmäßigen Abständen. Vom Bundesamt für Sicherheit in der Informationstechnik wurde die Entwicklung eines solchen Werkzeugs in Auftrag gegeben [11].

7 Zusammenfassung

In diesem Beitrag wurden ein Konzept und ein Prototyp vorgestellt, mit denen Java-2-Zugriffspolitiken einfacher verwaltet werden können. In einem Politikeintrag können nicht nur Zugriffsrechte gewährt werden, sondern es kann auch auf weitere Politikeinträge verwiesen werden. Die Politikeinträge werden entsprechend der dadurch definierten Baumstruktur ausgewertet. Politiken, Politikeinträge und Zugriffsrechte werden in einem LDAP-Verzeichnisdienst abgelegt. Durch die Zugriffsrechte des LDAP-Servers kann festgelegt werden, wer welche Teilpolitiken modifizieren kann. Damit kann die Verwaltung auch auf mehrere Administratoren aufgeteilt werden.

Literatur

1. Falk, R., Trommer, M.: Nutzung von Verzeichnisdiensten zur integrierten Verwaltung heterogener Sicherheitsmechanismen. In P. Horster (Hrsg.), Proceedings of Sicherheitsinfrastrukturen, Hamburg-Harburg, DuD-Fachbeiträge, Vieweg (1999), 96–108
2. Ferraiolo, D. F., Barkley, J. F., Kuhn, D. R.: A role based access control model and reference implementation within a corporate intranet. ACM Transactions on Information and System Security 1(2) (1999) 34–64
3. Freier, A., Karlton, P., Kocher, P.: The SSL Protocol Version 3.0. Netscape Corp. (1996), http://home.netscape.com/eng/ssl3/
4. Gong, L.: Java Security Architecture (JDK1.2). Sun Microsystems, Version 1.0 (1998), http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/security-spec.doc.html
5. Howes, T. A., Smith, M. C., Good, G. S.: Understanding and Deploying LDAP Directory Services. Macmillan Press, Basingstoke (1999)
6. Kassab, L. L., Greenwald, S. J.: Towards formalizing the Java security architecture of JDK 1.2. In Proceedings of European Symposium on Research in Computer Security (ESORICS'98), LNCS, Springer Verlag, Berlin (1998), http://www.gate.net/∼sjg6/publications/ESORICS1998.ps
7. Nikander, P., Partanen, J.: Distributed policy management for JDK 1.2. In Proceedings of the 1999 Network and Distributed System Security Symposium, San Diego, CA, ISOC (1999), http://www.isoc.org/isoc/conferences/ndss/99/proceedings/papers/nikander.pdf
8. OpenLDAP Foundation, OpenLDAP 1.2 (1999), http://www.openldap.org/
9. Rubin, A. D., Geer, D. E.: Mobile code security. IEEE Internet Computing 2(6) (1998) 30–34
10. Sandhu, R.: Role-based access control. Advances in Computers 46 (1998), http://www.list.gmu.edu/articles/advcom/a98rbac.pdf
11. Schwigon, H.: Möglichkeiten und Grenzen der Abwehr von Sicherheitsbedrohungen über Internet-Dienste durch Java-Applets. In Proceedings of 6. Deutscher IT-Sicherheitskongreß, 17.–19. Mai, Bonn, Bundesamt für Sicherheit in der Informationstechnik, SecuMedia Verlag, Ingelheim (1999), 71–75
12. Sun Microsystems, Java 2 SDK Documentation. Version 1.2.1 (1999), http://java.sun.com/products/jdk/1.2/docs/
13. Sun Microsystems, Java Naming and Directory Interface 1.2 beta (1999), http://java.sun.com/products/jndi/1.2/index.html