Policy-Beschreibungssprachen - Survey

Norbert Schmidbartl
Betreuer: Nadine Herold, M.Sc. Dipl.-Inf. Stephan Posselt
Seminar Innovative Internettechnologien und Mobilkommunikation WS1314
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: schmidba@in.tum.de

Kurzfassung

Um Abweichungen von einem gewünschten Systemzustand zu erkennen, können Monitore, zum Beispiel in Rechnernetzen, für Überwachungszwecke eingesetzt werden. Eine wichtige Frage, die sich bei der Verwendung dieser Monitore stellt, ist, wie mit Alarmen sinnvoll umgegangen werden kann. Speziell zur Lösung dieser Problemstellung eignet sich der Einsatz verschiedener Policy-Beschreibungssprachen, mithilfe derer man Bedingungen für Ereignisse (Alarmmeldungen) definieren kann, die dann automatisiert bestimmte Aktionen auslösen können. Dieses Paper beschäftigt sich mit dem Aufbau und der Eignung verschiedener Policy-Systeme und -Sprachen, beleuchtet Vor- und Nachteile, und vergleicht die Mächtigkeit dieser Sprachen. Es werden zudem aktuelle Lösungsansätze und Ideen vorgestellt, die sich mit möglichen Optimierungen der bestehenden Systeme beschäftigen.

Schlüsselworte

Policy, IDS, Monitor, Beschreibungssprache, Trigger

1. Einleitung

Um die Sicherheit und den reibungslosen Ablauf moderner IT-Systeme gewährleisten zu können, werden heutzutage in vielen Szenarien Intrusion-Detection-Systeme (IDS) und andere Monitore eingesetzt. Sobald eine Abweichung von einem gewünschten Systemzustand auftritt, wird eine entsprechende Meldung bzw. Alarm ausgegeben. Dadurch können unter Umständen sehr viele Meldungen generiert werden. Der Einsatz von Policy-Systemen hilft, angemessen, schnell und präzise auf diese Daten reagieren zu können. Im Vordergrund steht als Motivation eine weitgehende Automatisierung bestehender Vorgänge. Um dies zu ermöglichen, sollten bestimmte Rahmenbedingungen durch die Policy-Systeme erfüllt werden: Die genauen Umstände des Alarms sollten festgestellt werden können, und es sollte automatisiert eine Auswahl an adäquaten Reaktionen getroffen werden. Policies können demzufolge als Richtlinien betrachtet werden, nach denen automatisch geeignete Verhaltensweisen eines Systems ausgewählt werden. [2]

Die verschiedenen Policy-Sprachen unterscheiden sich in ihrer Struktur und Konzeption, weshalb sich die Frage stellt, in welchen Punkten die jeweiligen Beschreibungssprachen in Bezug auf Mächtigkeit und Eignung für IDS und andere Monitore Vorteile aufweisen können. Policy-Beschreibungssprachen sind ein mächtiges Werkzeug, mit deren Hilfe Aufgabenstellungen, die ansonsten manuell bearbeitet werden müssten, einheitlich, schnell und autonom gelöst werden können. Richtlinien und Bedingungen, die den Zustand des Gesamtsystems betreffen, sollten immer eingehalten werden. Es sollte eine Beschreibungssprache gefunden werden, die diese Einschränkungen beachtet, aber trotzdem mächtig genug ist, um damit flexible Richtlinien definieren zu können. Wichtige Eigenschaften, die eine Policy-Beschreibungssprache unterstützen sollte, sind somit [10]:

1. Konsistenz: Die Policies, die für ein System geschrieben wurden, sollten nicht in sich selbst (und anderen Policies gegenüber) widersprüchlich sein.
2. Präzision: Die Richtlinien, die durch die Policies ausgedrückt werden, sollten klar formuliert und eindeutig sein, also nur eine Interpretation zulassen.
3. Kompatibilität: Die Policies müssen mit den Eigenschaften und Funktionalitäten des Systems, für das sie geschrieben wurden, harmonieren.
4. intuitive Verwendbarkeit: die Policies sollten benutzerfreundlich und möglichst einfach verwaltet werden können.

Policie-Sprachen lassen sich zudem grundsätzlich zwei Kategorien (Level) zuordnen: Der Kategorie der Low-Level-Policies, die sich durch eine maschinelle Interpretier- und Verwendbarkeit auszeichnen, und der Kategorie der High-Level-Policies, die sich am intuitiven Verständnis der Menschen orientieren. High-Level-Policies können durch den Einsatz von Low-Level-Policies in Policy-Systemarchitekturen realisiert werden. Das Policy Management Tool kann zudem High-Level-Policies, die für Menschen leicht verständlich sind, in Low-Level-Policies übersetzen, die wiederum für die Verarbeitung durch Computer optimiert sind. [10]

Die folgenden Kapitel geben einen Überblick über bestehende Policy-Systeme und -Sprachen, und zeigen Einsatzmöglichkeiten, neue Entwicklungen sowie Lösungsansätze auf.

2. Policy-Systeme im Überblick

Um den konkreten Einsatz und den Nutzen von Policy-Beschreibungssprachen besser verstehen zu können, ist es wichtig, ein prinzipielles Verständnis dafür zu entwickeln, wie moderne Policy-Systeme aufgebaut sind, wie sich ein typischer Workflow in solch einer Umgebung gestaltet kann, und wie mithilfe dieser Policy-Systeme auf bestimmte Typen von Ereignissen angemessen reagiert werden kann. In den folgenden Kapiteln werden diese Fragestellungen näher beleuchtet.

doi: 10.2313/NET-2014-03-1_11

2.1 Policy-Architekturen

Es existieren mittlerweile Standards, an denen sich moderne Policy-Architekturen orientieren sollten. Einige dieser Vorgaben stammen von der IETF (Internet Engineering Task Force), die in den RFCs (Request for Comments) 2748 und 2753 näher beschrieben werden [10]. Ein Policy-System besteht demnach grundsätzlich aus vier Einheiten: den Policy Enforcement Points (PEP), einem Policy Decision Point (PDP), einem Policy Management Tool sowie aus mindestens einem Policy Repository (PR) [1]. Wie die einzelnen Komponenten zueinander in Relation stehen, kann Abbildung 1 entnommen werden.

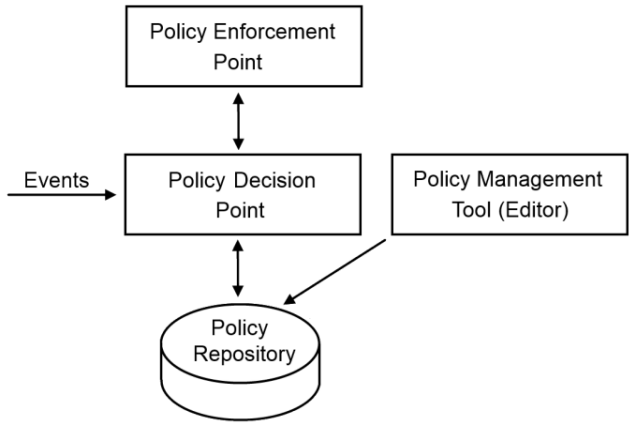

Abbildung 1. Policy Systemarchitektur (IETF) [1][2][10]

Die Komponenten übernehmen folgende Aufgaben [13]:

1. Policy Decision Point (PDP): Der PDP interpretiert die in den PRs gespeicherten Policies und schickt passende Instruktionen an den PEP.
2. Policy Enforcement Point (PEP): Der PEP führt die Anweisungen des PDP aus. Jedes Gerät des Systems besitzt einen eigenen PEP, dem genau ein PDP zugeordnet ist. In den Policies muss somit spezifiziert sein, welcher PEP zuständig ist.
3. Policy Management Tool: Administratoren können dieses Werkzeug nutzen, um Policies in das System einzupflegen. Das Policy Management Tool stellt somit eine Schnittstelle zwischen dem PR, in dem Policies gespeichert werden, und den Administratoren dar. Es existieren Policy Management Tools, die von Menschen generierte, leicht lesbare High-Level-Policies in maschinenlesbare Low-Level-Policies übersetzen können.
4. Policy Repository (PR): im PR werden alle Policies, die vom Policy Management Tool erzeugt werden, gespeichert. Sie können bei Bedarf vom PDP abgerufen werden.

Die Grundkomponenten der Policy-Systemarchitektur können für die Kommunikation untereinander unterschiedliche Protokolle verwenden, wie z.B. COPS, SNMP, HTTP oder telnet. Für die Kommunikation mit dem PR ist das LDAP-Protokoll (Lightweight Directory Access Protocol) ein weit verbreiteter Standard [2]. Die später vorgestellten Policy-Sprachen verwenden in ihren Implementierungen ähnliche Architekturen, in denen sich Einheiten analog zu den hier vorgestellten Komponenten finden lassen.

2.2 Workflow und mögliche Szenarien

Ein typischer Ablauf innerhalb eines Rechnernetzwerks kann sich beispielsweise wie folgt gestalten: Ein IDS oder ein anderer Monitor löst einen Alarm aus. Dies erfolgt, sobald ein verdächtiges Verhalten oder ein fehlerhafter Zustand festgestellt wurde. Durch statische Analysen, der Verwendung von Filtern und bekannten Angriffssignaturen sowie heuristischen Methoden können solche Angriffe und Fehlerzustände erkannt und anschließend gemeldet werden. Wie solch ein Alarm im Detail kommuniziert werden kann, unterscheidet sich von System zu System. Eine Meldung kann direkt, zum Beispiel per E-Mail, an einen Administrator geschickt werden, es kann aber auch ein Policy-System angesprochen werden, das anhand von gespeicherten Richtlinien automatisch entscheidet, wie weiter vorgegangen werden soll. Abbildung 2 soll die Zusammenhänge zwischen den einzelnen Komponenten veranschaulichen. [12]

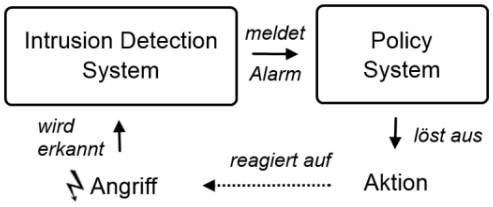

Abbildung 2. Typisches Szenario – Workflow

2.3 Umgang mit Alarmen

Sobald ein Ereignis einem Policy-System gemeldet wurde, sollten die vorhandenen Richtlinien automatisch eine geeignete Reaktion definieren, falls bestimmte Bedingungen hierfür erfüllt sind. In diesem Zusammenhang ist es wichtig zu wissen, dass IDS mit einer gewissen Wahrscheinlichkeit „false-positives“ (Fehlalarme) liefern können, also eine Meldung nicht unbedingt auf einem tatsächlichen Fehlerzustand oder Angriff beruhen muss. Das Policy-System kann nur dann optimal auf solche Fehlalarme reagieren, falls die genauen Umstände des Alarms dem Policy-System bekannt sind. Wichtige System– und Umgebungsvariablen sollten deshalb von den Policy-Systemen abrufbar sein, und es sollten die für die Meldung zuständigen Stellen, also Personen (Administratoren) oder Security Systeme, situationsbedingt ausgewählt und automatisch benachrichtigt werden können. Trotzdem sollte Wert darauf gelegt werden, dass die Privatheit sensibler Daten gewährleistet bleibt, und nicht beispielsweise durch das Einschleusen fehlerhafter Policies sensible Sicherheitslücken (z.B. durch fälschlicherweise geöffnete Ports) entstehen. Ein Policy-System, das komplett auf den Zugriff auf System- und Umgebungsvariablen verzichtet, kann jedoch nur sehr eingeschränkt und oberflächlich auf Angriffe reagieren. Bei der näheren Beschreibung und Beurteilung der in diesem Paper vorgestellten Policy-Beschreibungssprachen wird deshalb darauf geachtet, wie und in welchem Umfang entsprechende Systemparameter von den Policies im Bedingungsteil abgefragt werden können. [12]

2.4 Geeignete Reaktionen

Bestimmte Ereignisse können dazu führen, dass Policies, die im PR gespeichert sind, vom PDP und PEP ausgewertet werden.

doi: 10.2313/NET-2014-03-1_11

Diese Art von Ereignis wird nachfolgend als „Trigger“ bezeichnet. Das Ereignis, das die Auswertung anstößt, sollte in den Richtlinien klar definiert werden. Nachdem ein Trigger-Ereignis die Auswertung einer Policy angestoßen hat, und der Bedingungsteil erfüllt wird, sollte die automatische Ausführung einer Aktion erfolgen. Zu diesem Zeitpunkt wird davon ausgegangen, dass kein Fehlalarm vorliegt. Es sollte exakt auf den Fehlerzustand und eine mögliche Bedrohung reagiert werden. So können beispielsweise betroffene Verbindungen unterbrochen und fehlerhafte Daten geändert oder verworfen werden. Zudem könnte zum Beispiel eine angeschlossene Firewall den Umstanden entsprechend konfiguriert werden. In vielen Fällen ist es sinnvoll, den zuständigen Administrator über den eigentlichen Vorfall zu informieren, falls eine automatische Korrektur fehlschlägt, oder das Problem nicht komplett beseitigt werden konnte. [12]

3. Einordnung und Analyse verschiedener Policy-Sprachen

Nicht alle Policy-Beschreibungssprachen eignen sich für den Einsatz in den vorgestellten Szenarien. So sind die Ereignisse, die die Auswertung einer Policy anstoßen, teilweise vordefiniert und somit nicht flexibel genug. Zwingende Voraussetzung ist, dass falls ein Ereignisse (Trigger) eintritt, ein dazugehöriger Bedingungsteil der Policy zuverlässig ausgewertet wird. Die Policy-Systeme sollten verbindlich und umgehend auf Meldungen (Ereignisse) des IDS reagieren. Dies ist bei einigen Policy-Spezifikationen nicht möglich. [1]

Auf eine Gruppe von Policy-Sprachen, die diese Voraussetzungen größtenteils nicht vollständig erfüllen können, aber trotzdem im IDS- und Monitor-Umfeld Anwendungsgebiete als Ergänzung finden können (zum Beispiel im Bedingungsteil), sei an dieser Stelle hingewiesen: Es handelt sich um die sogenannten Traffic-Flow-Policy-Sprachen, die mittels Pattern-Matching beliebige Datenströme auswerten können. Beispiele hierfür wären SLR, PAX-PDL sowie PDL. [11]

Eine der verbreitetsten XML-Policy-Sprachen, XACML, kann nur auf eine bestimmte Art von Events reagieren, nämlich Zugriffsversuche, und ist für unser Szenario nicht geeignet. Entsprechende Erweiterungen sind denkbar, im Moment existieren aber dazu noch keine Implementierungen. [11]

In den nachfolgenden Kapiteln beschäftigt sich dieses Paper ausschließlich mit Policy-Beschreibungssprachen, die sich bei den vorgestellten Szenarien als Policy-Instanz einsetzen lassen.

3.1 Wichtige Kriterien und Eigenschaften

Geeignete Policies weisen im Allgemeinen einige gemeinsame Eigenschaften auf. Jedes der Systeme muss sich mit den folgenden Entitäten auseinandersetzen, um eine automatisierte Bearbeitung eingehender Alarme gewährleisten zu können. [1]

1. Ereignisse (Events) führen zur Auswertung bestimmter Policies. Systemänderungen wie zum Beispiel IDS-Alarme sowie Benutzereingaben können solche Trigger darstellen. Die Ereignis-Definitionen können meist kombiniert, negiert sowie chronologisch gereiht werden. Die separate Definition von Ereignissen dient der Optimierung, kann aber prinzipiell auch innerhalb des Bedingungsteils erfolgen.
2. Bedingungen (Conditions): Tritt ein Ereignis ein, werden Bedingungen ausgewertet, die an den Systemzustand und diversen Umgebungsvariablen gerichtet sein können. Die Auswertung des Bedingungsteils liefert meist die booleschen Werte „true“ oder „false“ zurück.
3. Aktionen (Actions): Ist die Bedingung erfüllt („true“), wird mindestens eine Aktion, die den Systemzustand ändert, automatisch ausgeführt.

Das Event-Condition-Action-Pattern (ECA) kann, wie in Abbildung 3 verbildlicht, in Kurzform folgendermaßen beschrieben werden:

ON (Event) IF (Condition) THEN (Action)

Eine weitere Grundstruktur vieler Policies sieht zudem wie folgt aus[1]:

IF (Condition) THEN (Action)

ECA stellt eine Optimierung dar: die Trigger (Ereignisse), die die Auswertung des Bedingungsteils (Condition) bewirken, können separat definiert werden. Der Bedingungsteil wird, im Vergleich zum IF-THEN-Pattern, erst dann ausgewertet, falls solch ein Ereignis tatsächlich eintritt, wodurch System-Ressourcen eingespart werden können [1]. Es sollte darauf geachtet werden, dass Ereignisse und Bedingungen trotzdem flexibel genug im ECA-Pattern definiert werden können, und mögliche Einschränkungen aufgrund der Event-Condition-Aufteilung nicht benötigte Funktionalitäten beeinträchtigen.

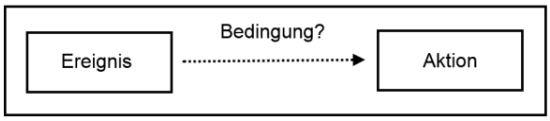

Abbildung 3: Event-Condition-Action

Im Gegensatz dazu muss bei der Verwendung einer IF-THEN-Struktur das Ereignis, dass die Ausführungen der Aktion auslöst, innerhalb des Bedingungsteils definiert werden. Es stellt sich somit die Frage, wann solch eine Policy ausgewertet werden soll. Einige Policy-Sprachen sehen eine Auswertung der Policies nur vor, falls ein ganz bestimmter Typ von Ereignis eintritt. Das wäre für den genannten Anwendungsfall zu unflexibel. Zudem wäre es möglich, bei jedem Ereignis die Conditions aller gespeicherter Policies auszuwerten, was gegenüber ECA bei umfangreichen Bedingungsteilen rechenaufwändiger wäre. Der Bedingungsteil der IF-THEN-Policies kann zudem standardmäßig in bestimmten Zeitintervallen ausgewertet werden, wodurch ein Zugriff auf alle Policies nicht mehr bei jedem einzelnen Event nötig ist, sondern die Menge aller zwischenzeitlich eingetretenen Events auf einmal geprüft werden kann. Dies erlaubt aber weder eine asynchrone Verarbeitung der Alarme, noch stellt es eine optimale Nutzung der vorhandenen Systemressourcen dar. Allerdings sind sowohl die ECA-Struktur als auch das IF-THEN-Pattern prinzipiell für die hier vorgestellten Szenarien geeignet, falls die Ereignisse mithilfe der Beschreibungssprache flexibel genug definiert werden können. [1]

Ein weiteres Kriterium, das die Verwendbarkeit von Policy-Beschreibungssprachen stark beeinflusst, ist die Flexibilität und Komplexität der Grammatik. Interessant ist hierbei, wie sich bestehende Definitionen kombinieren lassen. Für den Bedingungsteil ist es wichtig, dass flexibel und dennoch mit der nötigen Präzision definiert werden kann, welcher Zustand zu einer

doi: 10.2313/NET-2014-03-1_11

positiven Auswertung („true“) führt. Beschreibungen sollten im Idealfall durch Aggregation, durch Negation, durch die Verwendung von Ereignissequenzen und Gruppierungen auch für komplexere Systemzustände angepasst werden können. Dadurch erhöht sich die Wiederverwendbarkeit der Policies (Modularisierung).

3.2 Vergleich verschiedener Policy-Beschreibungssprachen

Im Folgenden werden verschiedene Policy-Beschreibungssprachen näher betrachtet, Vor- und Nachteile aufgezeigt sowie ihre Eignung für IDS und andere Monitore dargestellt.

3.2.1 PONDER

PONDER ist eine deklarative, objekt-orientierte Sprache, mit deren Hilfe man in erster Linie Policies für verteilte Systeme schreiben kann. Neben der Definition von Autorisierungs-Policies ist es möglich, Ereignis-getriggerte „Obligation Policies“ zu definieren [13]. Diese Art der Policies eignet sich hervorragend für unser Szenario. Es ist zudem möglich, Policies zu gruppieren, beziehungsweise in Relation zueinander zu setzen (Composite Policies) [2]. Die Grammatik von PONDER ist in EBNF (Erweiterte Backus-Naur-Form) beschrieben, und kann jederzeit eingesehen werden. [9]

Ein Vorteil, den die Obligation-Policies von PONDER bieten, ist die ECA–Struktur, die eine präzise, ereignisabhängige Definition von Policies erlaubt [1]. Es können externe Skripte und Programmteile ohne Probleme von PONDER importiert werden, der Funktionsumfang kann somit theoretisch beliebig erweitert werden. Wichtige Systemparameter und Umgebungsvariablen können von PONDER-Policies abgefragt und verarbeitet werden. [9]

Betrachten wir nun zunächst den grundsätzlichen Aufbau einer PONDER Obligation Policy [10]:

inst	**oblig**	*Policy-Titel*”**{**”
	on	*Event-Spezifikation*;
	subject [*Typ*]	*Domäne***;**
	[**target** [*Typ*]	*Domäne***;**]
	do	*Aktionsliste***;**
	[**catch**	*Ausnahmebehandlung***;**]
	[**when**	*Bedingungsteil***;**] ”**}**”

Das ECA-Konzept wird bei PONDER durch die Schlüsselwörter

on (Event) when (Condition) do (Action)

umgesetzt. Ein weiteres wichtiges Kriterium, das PONDER erfüllt, ist, dass Ereignisse präzise beschrieben werden können. Ein Ereignis kann grundsätzlich eintreten, sobald ein Basisereignis (Trigger) dem Policy-System gemeldet wird. Zudem können Ereignisbeschreibungen durch Operatoren kombiniert werden (Tabelle 1).

Tabelle 1. Event Compositions in PONDER [9]

Operator	Erklärung (Trigger)
e1 && e2	e1 und e2 treten auf
e + t	tritt nach einer Zeitspanne t nach e auf
{e1 ;e2}!e3	e1 tritt vor e2 auf, ohne e3 dazwischen
e1 | e2	e1 tritt vor e2 auf
n * e	e tritt genau n-mal auf

Nach dem Keyword „do“ kann in PONDER entweder eine einzelne Aktion, oder eine Komposition, bestehend aus mehreren Aktionen, beschrieben werden. Aktionen können sequenziell oder nebenläufig ausgeführt werden. Eine Aktion kann in PONDER importiert werden, und ist somit sehr flexibel. [10]

Falls eine Ausnahme auftritt, kann entsprechend reagiert werden (catch). Der Bedingungsteil (when) in PONDER ist ein Ausdruck in der Object Constraint Language Version 3 (OCL), und somit mächtig genug, komplexere Bedingungen zu beschreiben. Auf die Systemzeit kann ohne die Verwendung externer Skripte zugegriffen werden. Externe Ressourcen, Methoden und Systemvariablen können in PONDER verwendet werden.

Tabelle 2. Action Compositions in PONDER [9]

Operator	Erklärung (Ausführung)
a1 -> a2	a2 muss a1 folgen (sequenziell)
a1 || a2	a1 oder a2 muss terminieren (nebenläufig)
a1 && a2	a1 und a2 müssen terminieren (nebenläufig)
a1 | a2	falls a1 nicht ausgeführt werden kann: a2

Für PONDER existieren zahlreiche Implementierungen, Management-Tools sowie Dokumentationen. Zudem ist die Struktur des Policy-Systems dem IETF-Modell, das in Kapitel 3 vorgestellt wurde, relativ ähnlich. Der LDAP-Standard kann für die Kommunikation mit dem PR genutzt werden. [10]

Alles in allem lässt sich festhalten, dass sich PONDER bei Szenarien im Umfeld von IDS oder anderen Monitoren ohne bedeutende Einschränkungen gut einsetzen lässt. Zu PONDER gibt es zudem viele Implementierungen, und das Policy-System hat sich in bereits der Praxis bewährt.

3.2.2 Ponder2

Wenn man Syntax und Funktionsumfang von Ponder2 näher betrachtet, fällt auf, dass es mit dem als Vorgänger bezeichneten PONDER nicht mehr viel gemein hat. Ponder2-Policies sind in der sogenannten PonderTalk-Sprache verfasst [14]. PonderTalk orientiert sich stark an Smalltalk, einer objektorientierten Programmiersprache. Der Umstand, dass eine vollwertige, objektorientierte Sprache als Vorbild für PonderTalk genommen wurde, verdeutlicht, dass hier größten Wert auf den Funktionsumfang und die Mächtigkeit der Sprache gelegt wurde. Es können beliebige Daten, die von den Policies genutzt werden, in PonderXML (an XML angelehnt) gespeichert und organisiert werden, worauf dann zudem XPath-Anfragen möglich sind.

doi: 10.2313/NET-2014-03-1_11

Außerdem können sogenannte „Managed Objects“ (analog zu Smalltalk-Klassen) in Java geschrieben werden, die die Grundbausteine eines Ponder2-Systems bilden und Funktionalitäten für die Policies bereitstellen. [14]

Die Policies, die für unsere Szenarien interessant sind, werden auch in Ponder2 „Obligation Policies“ genannt, und es wird wieder explizit von einem ECA-Pattern gesprochen. Es können somit Events, Conditions und Actions separat in PonderTalk definiert werden. [14]

Eine Policy in Ponder2 kann beispielsweise wie folgt aussehen [15]:

Event in Ponder2:

```
template := root/factory/event create: #( "monitor" "value" ).
root/event at: "monitor" put: template.
```

Obligation Policy in Ponder2:

```
policy := root/factory/ecapolicy create.
policy event: root/event/monitor;
       condition: [ :value | value > 100 ];
       action: [ :monitor :value |
            root print: "Monitor " +
            monitor + " has value " + value
       ];
       active: true.
```

Aus dem Beispielcode ist ersichtlich, das XPath-ahnliche Anfragen den Typ der Policy bestimmen, sowie das Ereignis definieren. Condition- und Action-Teil sind in Ponder2 in der PonderTalk-Syntax beschrieben. Die ECA – Eigenschaften der Ponder2-Richtlinien können somit in PonderXML und PonderTalk festgelegt werden. Zudem können Ponder2-Obligation-Policies aus XML-Dateien erstellt werden [15]. Durch die Verwendung von verbreiteten Standards und sehr umfangreichen Sprachen wie PonderTalk und Java sind dem Zugriff auf Systemressourcen prinzipiell wenig Grenzen gesetzt. Im Rahmen des ECA-Patterns können Ressourcen des Systems abgerufen und abgefragt werden. Allerdings ist die Kombination von Events im Vergleich zu PONDER etwas aufwändiger, es muss auf Workarounds zurückgegriffen werden. [14]

Durch die Komplexität des Systems und die vielen verwendeten Sprachen (PonderTalk, Java, PonderXML) erhöht sich allerdings der Aufwand, der insgesamt für die Verwaltung eines Ponder2-System gegenüber PONDER nötig ist. Wenn man allerdings den Funktionsumfang und Mächtigkeit betrachtet, ist Ponder2 bestens für eine Verwendung im IDS- und Monitorumfeld geeignet.

3.2.3 PDL/PPDL

Policies, die in der Policy Description Language (PDL), beziehungsweise deren Erweiterung PPDL (Preferential PDL) geschrieben sind, verwenden, ähnlich wie PONDER, die ECA-Struktur zur Formulierung ihrer Richtlinien. Die PDL-Spezifikation sieht grundsätzlich folgenden Aufbau der Policies vor[1] [3][6]:

(Event) causes (Action) if (Condition)

Eine Besonderheit von PPDL ist hierbei, dass Policies der Form

true causes (Action) if (Condition)

erlaubt sind, was die Policy auf eine simple IF-THEN-Form reduzieren würde [7]. PPDL erweitert PDL zudem um globale Richtlinien der Form

never (Action 1, Action 2, …, Action n) if (Condition)

die eine simultane Ausführung von Action 1, Action 2 , …, Action n verhindert, sobald die Bedingung (Condition) erfüllt ist. Diese Aktionen dürfen dann nach dieser Richtlinie nur sequenziell ausgeführt werden. Dies ist sinnvoll, wenn die Aktionen auf gemeinsame Ressourcen zugreifen möchten. [7]

Es wird darauf hingewiesen, dass die Auswertung einer Policy, ähnlich wie bei PONDER, auch durch eine Kombination mehrerer (Basis-)Events ausgelöst werden kann. Es wird zwischen Basisevents E und komplexen Events (E) unterschieden. Events können durch UND- (&) beziehungsweise ODER (|)- Operatoren verknüpft werden, sequentiell (E1, E2, E3, E4) oder als Wiederholung eines einzelnen Basisevents definiert werden (^E). Bedingungen bestehen aus mehreren Teilbedingungen. Diese umfassen Prädikaten, die durch gängige Operatoren in Relation gesetzt werden können (=, !=, <, <=, >, >=). [3]

Eine weitere Besonderheit von PPDL ist, dass die Auswertung von Policies, die durch Ereignisse ausgelöst werden, wiederum neue Ereignismeldungen (Trigger) auslösen können. Dies geschieht durch das Schlüsselwort „trigger“:

(Event 1) triggers (Event 2) causes (Action) if (Condition)

Die Wiederverwendbarkeit von Policies kann mithilfe solcher Konstrukte durch Modularisierung deutlich erhöht werden. [7]

PPDL kann zudem theoretisch auf beliebige Systemparameter oder primitive Systemfunktionen im Rahmen des ECA-Patterns zugreifen, womit die Umstände eines Alarms von einer PPDL-Policy jederzeit geprüft werden können. [3] [7]

Die Autoren von PDL/ PPDL gehen einen anderen Weg als die Schaffer von PONDER. Sie definieren zuerst die High-Level-Syntax von PDL/PPDL. Um dann jedoch eine maschinelle Interpretierbarkeit der Policy-Semantik zu erreichen, wird erläutert, wie eine PDL-Policy in PROLOG (Programmation en Logique) [6], beziehungsweise eine PPDL-Policy in LPOD (Logic Programs with Ordered Disjunctions) oder ASP (Answer Set Programming)- Code übersetzt werden kann [7]. Es existieren zudem Shell-Skripte, die diesen Code dann ausführen können, außerdem ist ein Übersetzer von PPDL in LPOD in Arbeit. In Zukunft soll die Präzision, sowie die Konsistenz von PDL weiter erhöht werden. [8]

Im Vergleich zu PONDER gibt es weniger ausgereifte Implementierungen, und der Funktionsumfang von PPDL ist geringer. In der Theorie jedoch ist PPDL als Policy Beschreibungssprache im Umfeld von IDS und andere Monitoren mit dem ECA-Konzept gut geeignet. [8]

3.2.4 SPL

Die Security Policy Language (SPL) wurde in der Version 3.0 zuletzt 2007 spezifiziert, ist also eine im Vergleich zu den anderen Policy-Beschreibungssprachen eine relativ junge Sprache. Dies wird deutlich, wenn man die Auszeichnungssprache betrachtet, in der SPL-Policies verfasst werden: XML. Dies kann in Hinblick auf die fortschreitende Standardisierung gewisser Richtlinien als langfristiger Vorteil gegenüber anderen Sprachen betrachtet werden. [4]

doi: 10.2313/NET-2014-03-1_11

SPL wird durch das POSITIF Framework unterstützt, und umfasst ähnlich wie PONDER eine Reihe von unterschiedlichen Policy-Typen (Authentication, Authorization, Filtering, Channel Protection sowie Operational Policies). [4]

Das XML-Format von SPL lässt eine hierarchische, standardisierte Beschreibung der einzelnen Elemente der Policy-Spezifikation zu. Es können Rollen verteilt, Privilegien vergeben und Filter-Regeln gesetzt werden, die den Netzwerk-Traffic überwachen sollen. [4]

Alle Policies der SPL enthalten folgende Eigenschaften im XML-Format: Name der Policy, Beschreibung der Policy in natürlicher Sprache, Richtlinien wie die Policy genutzt werden soll (formlos), eine Enabled-Eigenschaft, die die Policy aktiviert, und eine ValidityTimePeriod-Eigenschaft, die beschreibt, in welchem Zeitintervall die Richtlinien, die durch die Policy beschreiben werden, aktiv sind. Einmal ausgeführte Aktionen können dadurch allerdings nicht wieder automatisch rückgängig gemacht werden. [4]

Die Operational-Policies haben folgenden Aufbau:

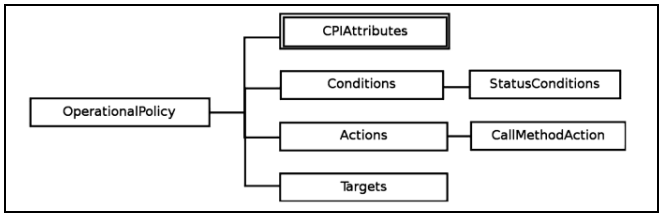

Abbildung 4. Operational Policy [4]

Die Condition-Eigenschaft beinhaltet die Bedingungen für die eigentliche Auswertung der Policy. Im Moment kann allerdings nur der Status von einem System-Element abgefragt werden. Da die Alarme theoretisch auch über ein System-Element übergeben werden können, wäre eine Verwendung in einem IDS-Szenario denkbar. Events, also Trigger lassen sich nicht explizit definieren. Das Action-Element kann eine beliebige Methode inklusive Parameter enthalten, es kann somit theoretisch jede denkbare Funktion als Aktion automatisch ausgeführt werden. Um die Policies strukturieren zu können existieren Policy Groups. [4]

SPL erweckt ähnlich wie PPDL einen etwas unfertigen Eindruck. Trotzdem ist ein Einsatz zumindest in der Theorie denkbar. Die Verwendung des XML-Formats erlaubt hierbei eine präzise, eindeutige und leicht verständliche Beschreibung diverser Richtlinien. [4]

3.2.5 CIM-SPL

Die Common Information Model Simplified Policy Language (CIM-SPL) basiert, wie die Abkürzung erkennen lässt, auf dem Common Information Model (CIM). Die DMTF Policy Working Group, die diese Spezifikation geschaffen hat, betont, dass es sich hierbei um eine

IF (Condition) THEN (Action)

Beschreibungssprache handelt. Die eigentlichen Trigger sind bei diesem Schema implizit in der Condition enthalten. [1] [5]

CIM-SPL wurde laut der DMTF Policy Working Group von PONDER sowie PDL beeinflusst. So können wie bei den Vorbildern Policies gruppiert werden. Außerdem ist eine Verschachtelung möglich. Auf diese Weise können Policy-Hierarchien aufgebaut werden. [5]

CIM-SPL verwendet wie PONDER und PPDL nicht die XML-Auszeichnungssprache. Die Grundstruktur einer Policy unter CIM-SPL sieht meist wie folgt aus [5]:

```
Policy {
  Declaration {
    <List of constant definition> (Optional)
    <List of macro definitions> (Optional)
   }
   Condition {       (Optional)
        <If Condition>
   }
   Decision {        (Required)
        <Then Decision>
   }
}: Priority
Condition {
    <Boolean Expression> (An expression that results
    in a Boolean constant after evaluation)
}
Decision {
    <action block>
}
```

Condition enthält die Bedingungen (als booleschen Ausdruck), die erfüllt sein müssen, um eine Entscheidung (Decision) zu treffen. Decision enthält die Action-Beschreibungen, die automatisch ausgeführt werden, falls die Bedingungen erüllt sind. Für die Beschreibung der Richtlinie und der Bedingungen stehen numerische, boolsche sowie relationale Operatoren zur Verfügung. Außerdem wird eine Vielzahl an Zeichenkettenoperationen nativ angeboten. Eine ganze Reihe weiterer Funktionen vereinfachen die Verwendung von CIM-SPL und erlauben die Abfrage von Systemzuständen. Bedingungen können ähnlich wie bei PONDER kombiniert werden. [5]

Es lässt sich sagen, dass CIM-SPL ein umfangreiches Spektrum an Anwendungsmöglichkeiten bietet. Die vielen Funktionalitäten, die direkt in CIM-SPL angeboten werden, und nicht erst importiert werden müssen, erhöhren die Konsitenz und die Stabilität. Die Verwendbarkeit von CIM-SPL nimmt dadurch zu. [5]

3.3 Vergleich der Mächtigkeit der Sprachen

Mithilfe der vorgestellten Sprachen lassen sich Richtlinien definieren, die präzise beschreiben, wie ein System angemessen auf einen Alarm reagieren sollte. Allerdings unterscheiden sich die Sprachen in ihrer Komplexität und Flexibilität. Einen Überblick über die Mächtigkeit der in diesem Paper vorgestellten Sprachen kann der Tabelle 3 entnommen werden. Es wird angegeben (++,+,o,-,--), wie umfangreich die Funktionalitäten der jeweiligen Sprache in den beschriebenen Bereichen (Auszeichnungssprache, ECA, Verwendbarkeit) relativ zu den anderen Beschreibungssprachen sind. Falls Implementierungen noch unvollständig, oder noch nicht vorhanden sind (Stand 2013), oder sich das System in der jetzigen Form nur mit diversen Workarounds für das IDS-Szenario eignet, ist dies durch ein „tba“ gekennzeichnet. Eine Sprache erfüllt das ECA-Kriterium nur, falls sie ausdrücklich als solche von den Autoren gekennzeichnet wurde. [1][3][4][7][11][13]

doi: 10.2313/NET-2014-03-1_11

Alle in diesem Paper vorgestellten Policy-Beschreibungssprachen lassen sich zumindest theoretisch in Verbindung mit IDS nutzen. Es bleibt festzuhalten dass sich die hier vorgestellten Sprachen teilweise stark in ihrer tatsächlichen Verwendbarkeit unterscheiden. PONDER, Ponder2 und die auf CIM aufbauenden Sprachen haben sich in der Praxis bewährt. [1]

Tabelle 3. Policy-Sprachen im Vergleich

	PON.	**Ponder2**	**PPDL**	**SPL**	**CIM-SPL**
ECA	Ja	Ja	Ja	Nein	Nein
Implementierungen	Ja	Ja	tba	Ja	Ja
IDS	Ja	Ja	Ja	tba	Ja
XML	-	o	-	+	-
Events	++	+	++	-	o
Condition	+	++	+	o	++
Action	++	++	++	+	++
Verwendbarkeit	+	+	o	o	+

PONDER eignet sich tendenziell für Systeme mit zentralem Rechner, die LDAP oder CIM verwenden. Die Verwendung von CIM-SPL ist vor allem dann sinnvoll, falls aufbauend auf CIM ein Policy-System für IDS konstruiert werden soll. Ponder2 weist hingegen darauf hin, dass es vor allem für hochskalierende, verteilte Systeme gedacht ist. Zudem ist Ponder2 nicht von LDAP oder CIM abhängig. Für größere Projekte mit höherem Verwaltungsaufwand eignet sich deshalb, laut Angabe der Autoren, Ponder2 besonders gut[14]. Für PPDL und SPL fehlen noch ausgereifte Implementierungen und Praxistests im Umfeld von IDS und anderen Monitoren. [1] [8][13]

Falls verwendbare Implementierungen vorhanden sind, wie beispielsweise bei PONDER der Fall, können die Policy-Beschreibungssprachen überall da eingesetzt werden, wo nach strikten Richtlinien automatisch auf Ereignisse reagiert werden muss. In verteilten System und diversen Rechnernetzen werden heutzutage in den meisten Fällen Monitore und IDS eingesetzt, die den Ablauf der Programme überwachen, und die durch Policy-Systeme sinnvoll ergänzt werden können. Ein Intrusion-Detection-System kann so ergänzt werden, dass es vollautomatisch die wichtigsten Sicherheitsmechanismen des verteilten Systems aktivieren und wichtige Systemressourcen schützen kann. Im Zusammenspiel mit anderen Sicherheitskomponenten bilden diese IDS in vielen Rechnernetzen mittlerweile den Kern der eigentlichen IT Sicherheit. [12]

4. Zusammenfassung

Die Verwendung von Policy-Beschreibungssprachen im Umfeld von IDS und anderen Monitoren ist besonders nützlich, wenn automatisch auf Alarme reagiert werden muss. Der Funktionsumfang und die Verwendbarkeit der unterschiedlichen Policy-Beschreibungssprachen können hierbei deutlich variieren, weshalb vor Einsatz eines neuen Policy-Systems geprüft werden sollte, welche Anforderungen an die Beschreibungssprache gestellt werden, ob einfache Lösungen vorhanden sind, oder ob zumindest auf Workarounds zurückgegriffen werden kann. Wenn man das Spektrum verfügbarere Policy-Sprachen als Ganzes betrachtet, fällt auf, dass zu einigen Beschreibungssprachen bisher nur unvollständige Implementierungen und rein theoretische Ansätze zu finden sind. Viele neue Ideen, fortschreitende Standardisierungen und verbreitete Policies wie PONDER[1], die ständig weiterentwickelt werden, machen das Universum der Policy-Beschreibungssprachen zu einem sehr dynamischen und fortschrittlichen wissenschaftlichen Terrain. Es bleibt abzuwarten, welche Policy-Beschreibungssprachen sich letztendlich in der Praxis langfristig bewähren werden. [1][3]

5. Literatur

[1] Weili Han, Chang Lei, Software School, Fudan University, Shanghai (2012), China. A survey on policy languages in network and security management.

[2] Alexander Keller, Heiko Ludwig (2004), Policy-basiertes Management: State-of-the-Art und zukünftige Fragestellungen

[3] Vitalian Danciu (2003), Entwicklung einer policy-basierten Managementanwendung für das prozeßorientierte Abrechnungsmanagement

[4] Positif.org (2007) Security Policy Language (SPL) - User Manual

[5] Distributed Management Task Force (2009), CIM Simplified Policy Language (CIM-SPL)

[6] Jorge Lobo, Dandeep Bhatia, Ahmim Naqvi (1999), A Policy Description Language

[7] Elisa Bertino, Alessandra Mileo (2005), PDL with Preferences

[8] Prof. Elisa Bertino (2004), PPDL: Preferential Policy Description Language, http://mag.dsi.unimi.it/PPDL/, zugegriffen: 10.12.2013

[9] Nicodemos Damianou, Naranker Dulay, Emil Lupu, Morris Sloman, Imperial College Research Report (2000), Ponder: A Language for Specifying Security and Management Policies for Distributed Systems

[10] Patricia Marcu (2005), Reference Installation of the Ponder Policy Toolkit

[11] Mohamed al-Morsy, Hossam M. Faheema (2009), A new standard security policy language

[12] BSI, Bundesamt für Sicherheit in der Informationstechnik, Intrusion-Detection-Systeme (IDS) (2013), http://l.hh.de/obnQZs, zugegriffen: 01.12.2013

[13] Matthias Ebert (2006), Konzeption und Implementierung einer policy-basierten Privacy Management Architektur für föderierte Identitätsmanagementsysteme am Beispiel Shibboleth

[14] Kevin Twidle, Naranker Dulay, Emil Lupu and Morris Sloman (2009), Ponder2: A Policy System for Autonomous Pervasive Environments

[15] Kevin Twidle (2008), ObligationPolicies, http://ponder2.net/cgi-bin/moin.cgi/ObligationPolicies, zugegriffen: 18.12.2013