Beveiliging is een cruciaal onderdeel van vrijwel ieder geautomatiseerd systeem. Het lijkt steeds belangrijker te worden; denk aan cloud toepassingen of het EPD. Beveiliging is een breed domein binnen ons vakgebied en beslaat grofweg twee deelgebieden: authenticatie en autorisatie. Dit artikel gaat dieper in op de autorisatie: wat mag iemand zien of doen?

XACML als standaard voor autorisatie

Rémon Sinnema is Principal Software Engineer bij EMC waar hij zich bezig houdt met cloudplatformen. Volg hem op Twitter (@sinnema313).

Voor autorisatie zijn verschillende modellen in omloop. De meest gebruikte zijn Access Control Lists (ACL's) en Role Based Access Control (RBAC). In ACL's zijn permissies opgeslagen bij een object. Bij RBAC krijgt een gebruiker een of meerdere rollen toegewezen, waaraan permissies worden gekoppeld. Beide modellen hebben zo hun nadelen. ACL's kennen weinig hergebruik en zijn dus niet efficiënt genoeg in scenario's met hele grote aantallen gegevens. Rollen zijn vaak niet fijnmazig genoeg om subtiele autorisatieregels te implementeren.

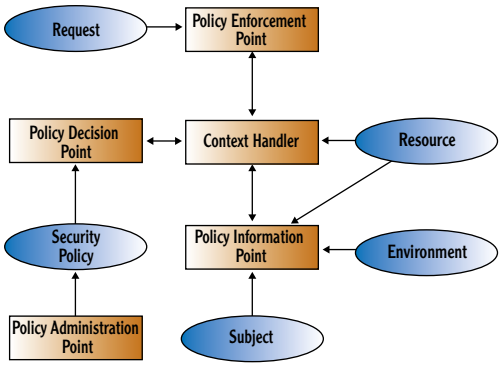
Vanwege deze nadelen worden tegenwoordig steeds vaker Attribute Based Access Control (ABAC) oplossingen toegepast. ABAC gebruikt attributen als bouwstenen in een gestructureerde taal om autorisatieregels vast te leggen. Attributen zijn naam-waarde paren en kunnen alle entiteiten beschrijven die voor autorisatie van belang zijn. In die zin is ABAC een generalisatie van zowel ACL als RBAC. De entiteiten zijn:

- Subject: degene die toegang wil (persoon, rol, applicatie).
- Resource: datgene waartoe de Subject toegang wil (web service, document, rij/cel in een database tabel).
- Action: datgene wat de Subject met de Resource doen wil (CRUD, GET/PUT/POST/DELETE)
- Environment: de context waarin de Subject de Action op de Resource doen wil (tijd, locatie, apparaat).

XACML

De OASIS standaard [1] voor het specificeren van ABAC-beleidsregels is eXtensible Access Control Markup Language (XACML) [2]. De regels in XACML zijn onafhankelijk van een bepaalde technologie en gescheiden van de gegevens waarop ze betrekking hebben, zodat de regels her te gebruiken zijn. Uiteraard werkt XACML goed samen met standaarden voor authenticatie, zoals Security Assertion Markup Language (SAML) [3].

Figuur 1 toont de componenten (oranje rechthoeken) in een XACML-systeem en de gegevens (blauwe ovalen) die tussen deze componenten stromen:



Figuur 1: Architectuur van een XACML-systeem.

Autorisatie in een op XACML gebaseerd systeem verloopt als volgt:

1. Een autorisatieverzoek (Request) komt binnen bij de Policy Enforcement Point (PEP);
2. De PEP stuurt het Request door naar de Context Handler
3. De Context Handler verzoekt de Policy Information Point (PIP) om additionele context attributen
4. De PIP verzamelt context attributen van het Subject (bijvoorbeeld diens rollen), de Resource (bijvoorbeeld diens locatie op schijf) en de Environment (bijvoorbeeld de huidige tijd) en geeft die terug aan de Context Handler
5. De Context Handler vraagt eventueel de inhoud van de Resource op
6. De Context Handler breidt het Request uit met de context attributen en eventueel de inhoud van de

Resource en biedt het uitgebreide Request aan de Policy Decision Point (PDP) aan
7. De PDP neemt een beslissing op basis van de autorisatieregels die via de Policy Administration Point (PAP) aangeleverd zijn
8. De PDP geeft de beslissing aan de Context Handler terug, die het op zijn beurt aan de PEP teruggeeft
9. De PEP geeft wel of geen toegang tot de Resource op basis van de beslissing van de PDP.

Doordat de componenten in een XACML-systeem gestandaardiseerd zijn, is het mogelijk om sommige componenten herbruikbaar te maken. Zo zullen we verderop in dit artikel een open source implementatie van de PDP behandelen. Andere componenten, zoals de PEP, zijn specifiek voor een applicatie.

Autorisatieregels in XACML

Autorisatiebeleid in XACML bestaat uit een hiërarchie van PolicySets, Policies en Rules. Een Rule combineert een Target, Effect en Condition. De Target beschrijft waarop de Rule van toepassing is (attributen van Subject, Resource, Action en Environment), het Effect geeft aan of de Rule toegang toestaat of verbiedt en de Condition verfijnt eventueel de Target verder. In de Condition kunnen functies gebruikt worden, zodat een zeer flexibel systeem ontstaat, dat zelfs de subtielste beveiligingsregels kan vastleggen. Rules zijn de kleinste herbruikbare stukjes autorisatiebeleid. Ze staan niet op zichzelf, maar zijn altijd onderdeel van een Policy.

Een Policy combineert een of meerdere Rules. Het heeft zelf ook een Target, die wordt gecombineerd met de Targets van de Rules om te zien of een Rule van toepassing is op een bepaald Request. Het Rule-Combining Algorithm van een Policy legt vast hoe de Effecten van meerdere toepasbare Rules worden gecombineerd. Obligations geven aan welke acties de PEP uit moet voeren. Een voorbeeld is dat de toegang gelogd moet worden in een audit trail.

Een PolicySet combineert meerdere Policies. Ook hier zien we een Target en Obligations. Een Policy-Combining Algorithm zorgt ervoor dat de resultaten van meerdere toepasbare Policies worden gecombineerd tot een eindresultaat. PolicySets zijn optioneel.

Zie [4] voor meer informatie over XACML en [5] voor enkele realistische voorbeelden van autorisatiebeleid.

XACML is zeer relevant voor de Java-ontwikkelaar. In de eerste plaats, omdat veel Javanen met webservers werken en die steeds vaker XACML ondersteunen, zoals Websphere [6] en WebLogic [7]. Maar belangrijker is dat met XACML autorisatieregels losgekoppeld worden van code. De ontwikkelaar kan zich meer richten op het schrijven van business logica, alleen de PEP hoeft nog ontwikkeld te worden. Dit in tegenstelling tot bijvoorbeeld Spring Security [8], waar de ontwikkelaar meer met autorisatie bezig moet zijn dan hem vaak lief is. (Overigens is men bezig met een PEP implementatie voor Spring Security [9].)

Om XACML te gebruiken in een Java-applicatie, moeten we zoals gezegd een PEP ontwikkelen, die een PDP aanroept. Dat is eenvoudiger dan het misschien klinkt: we hoeven alleen maar een Request te vullen met attributen, dat aan een PDP te geven en toegang toe te staan of te weigeren aan de hand van het antwoord van de PDP. De PDP hoeven we zelf niet te schrijven, die kunnen we hergebruiken. Er is bijvoorbeeld een open source implementatie van Oracle/Sun [10]. Zie [11] voor hoe deze PDP te gebruiken en hoe een PEP te ontwikkelen die deze PDP aanroept.

Als je tegen de Sun XACML API programmeert, ben je daar natuurlijk wel aan gekoppeld. Dat maakt het lastiger om van PDP te wisselen. De standaard manier om dat te voorkomen, is het schrijven van een façade [12]. Gelukkig hoef je ook dit niet zelf te doen, want voor Java wordt een dergelijke façade al geleverd door het OpenAz project [13]. Hun interface is wat moeilijker dan misschien nodig door het overvloedig gebruik van generics, maar je voorkomt er wel vendor lock-in mee.

Conclusie

We kunnen het ons niet veroorloven om het beveiligingswiel steeds maar opnieuw te blijven uitvinden. We moeten naar standaarden toe die hergebruik toestaan, niet alleen van beveiligingscomponenten in software, maar ook van de beveiligingsregels zelf. XACML is die standaard voor autorisatie. «

We kunnen het beveiligingswiel niet steeds opnieuw blijven uitvinden.

Referenties

[1] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
[2] http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
[3] http://en.wikipedia.org/wiki/Security_Assertion_Markup_Language
[4] https://community.emc.com/docs/DOC-7314
[5] https://community.emc.com/docs/DOC-7410
[6] http://www.ibm.com/developerworks/websphere/library/techarticles/0705_orchard/0705_orchard.html
[7] http://download.oracle.com/docs/cd/E13222_01/wls/docs92/secwlres/xacmlusing.html
[8] http://static.springsource.org/spring-security/site/
[9] http://forum.herasaf.org/index.php?topic=129.0
[10] http://sunxacml.sourceforge.net/
[11] https://community.emc.com/docs/DOC-8672
[12] http://nl.wikipedia.org/wiki/Fa%C3%A7ade_%28informatica%29
[13] http://openliberty.org/wiki/index.php/OpenAz_Main_Page