JEUS Security 안내서

Document info

Document name: JEUS Security 안내서

Document date: 2005-06-06

Manual release version: 3

Software Version: JEUS 5

차 례

그림 목차

표 목차

매뉴얼에 대해서

매뉴얼의 대상

본 매뉴얼은 JEUS 5 보안과 관련된 사항 - 설정법, 작동 방법, 커스터마이징 방법 - 에 관해 자세히 설명하고 있다. JEUS 시스템 관리자는 반드시 읽어야 한다.

매뉴얼의 전제 조건

본 문서을 읽기 전에, JEUS Server 전반에 대해 기본적인 이해가 필요하다. JEUS Server 안내서를 읽는 것이 도움이 된다.

또한 JEUS 보안 아키텍쳐, J2SE 보안 아키텍쳐, 일반적인 보안 기술 - SSL/TLS, 디지털 인증, role-based 사용자 인증 모델 - 에 대한 기본적인 이해도 필요하다.

매뉴얼의 구성

본 문서는 다음과 같은 11 개의 주요 장으로 구성되어 있다.

1. 소개
2. 보안 시스템의 개요: JEUS 보안 시스템에 대한 소개와 개괄
3. 보안 시스템의 설정: 보안 도메인, 보안 서비스, Subject, Policy 등과 같은 주요 보안 요소를 설정하는 방법
4. 어플리케이션과 모듈에서 보안 설정: JEUS 보안 시스템에서 J2EE application, EJB, 웹 모듈에 보안을 설정하는 방법
5. 보안 시스템 운영: 런타임시 보안 시스템을 관리하는 방법
6. 보안 시스템 튜닝: 최고의 성능을 위해 보안 시스템을 튜닝하는 방법
7. 보안 수준 향상: 보안 시스템과 JEUS 에서 전반적인 보안 수준을 향상시키는 방법

8. 보안 시스템 API 프로그래밍: security system API 를 사용하는 application(예를 들면 Servlet)을 개발하는 방법
9. 커스텀 보안 서비스 개발하기: JEUS 에서 Customized Security Provider 를 개발하는 방법
10. JACC Provider: JEUS 보안 시스템에 JACC provider 를 개발해서 통합하는 방법
11. 결론: 마치는 말

또한 다음과 같은 10 개의 부록이 있다.

A SecurityAdmin 툴 사용: 보안 관리를 위한 명령어 라인 툴에 대한 설명

B base64 커맨드 라인 툴 레퍼런스: Base64 encoder/decoder 명령어 라인 툴에 대한 설명

C master.xml 과 slave.xml: 보안 서비스를 설정하는 파일에 대한 설명

D subjects.xml: XML 기반의 Subject 저장소(store)에 대한 설명 (user store).

E policies.xml: XML 기반의 Policy 저장소에 대한 설명

F 기본 SPI 구현 레퍼런스: JEUS 가 제공하는 디폴트 보안 서비스에 대한 설명

G JEUS Server Permissions 레퍼런스: JEUS Server 에서 체크되는 Permission 에 대한 설명

H 보안 이벤트 레퍼런스: 보안 시스템에 감지되는 표준 Security 이벤트에 대한 설명

I 보안 시스템 속성 레퍼런스: 보안 시스템과 관련있는 표준 시스템 속성에 대한 설명

J Security API&SPI Reference: 보안 시스템 관련 API 와 SPI 에 대한 Javadoc

관련 매뉴얼

다음에 소개하는 문서들은 본 매뉴얼과 관련된 문서이다.

- **JEUS Server 안내서**: JEUS 의 기본적인 설정 디렉토리 구조와 JEUSMain.xml 에 대한 정보를 제공.
- **JEUS Enterprise JavaBeans 안내서**: EJB 보안과 CSI 에 대한 정보 제공.
- **JEUS Web Container 안내서**: Servlet 보안과 SSL 리스너에 대한 정보 제공
- **Java Authorization Contract for Containers Specification Version 1.0**: JACC provider 에 대한 정보 제공
- **J2EE 1.4 Specification**: JEUS 를 포함한 J2EE 서버에 적용되는 기본적인 보안 아키텍쳐에 대한 정보 제공
- **EJB 2.1 Specification**: EJB 보안 모델에 대한 정보 제공
- **Servlet 2.4 Specification**: Servlet 보안 모델에 대한 정보 제공
- **Javadoc for J2SE 1.4 packages java.security, javax.security.auth and J2EE 1.4 package javax.security.jacc**: J2SE/J2EE 보안과 관련된 기본적인 클래스에 대한 상세한 정보 제공

일러두기

표기 예	내용
텍스트	본문, 12 포인트, 바탕체 Times New Roman
텍스트	본문 강조
CTRL+C	CTRL 와 동시에 C 를 누름
`public class myClass { }`	Java 코드
`<system-config>`	XML 문서

표기 예	내용
참조: / 주의:	참조 사항과 주의할 사항
Configuration 메뉴를 연다	GUI 의 버튼 같은 컴포넌트
JEUS_HOME	JEUS 가 실제로 설치된 디렉토리 예)c:\jeus
`jeusadmin nodename`	콘솔 명령어와 문법
[파라미터]	옵션 파라미터
< xyz >	'<'와 '>' 사이의 내용이 실제 값으로 변경됨. 예)<node name>은 실제 hostname 으로 변경해서 사용
|	선택 사항. 예) A|B: A 나 B 중 하나
…	파라미터 등이 반복되어서 나옴
?, +, *	보통 XML 문서에 각각 "없거나, 한 번", "한 번 이상", "없거나, 여러 번"을 나타낸다.
. . .	XML 이나 코드 등의 생략
<<FileName.ext>>	코드의 파일명
그림 1.	그림 이름이나 표 이름

OS 에 대해서

본 문서에서는 모든 예제와 환경 설정을 Microsoft Windows™의 스타일을 따랐다. 유닉스같이 다른 환경에서 작업하는 사람은 몇 가지 사항만 고려하면 별

무리 없이 사용할 수 있다. 대표적인 것이 디렉토리의 구분자인데, Windows 스타일인 “\”를 유닉스 스타일인 “/”로 바꿔서 사용하면 무리가 없다. 이외에 환경 변수도 유닉스 스타일로 변경해서 사용하면 된다.

그러나 Java 표준을 고려해서 문서를 작성했기 때문에, 대부분의 내용은 동일하게 적용된다.

용어 설명

다음 용어는 매뉴얼 전체에 걸쳐 자주 사용된다. 이해하기 어렵거나, 그 뜻이 모호한 용어에 대해 다음 목록을 참고하기 바란다.

Term	Definition
Anonymous Subject	"any one" 과 "no one" 를 나타내는 특별한 종류의 Subject. 익명 Subject 는 사용자 인증을 거치지 않는다.
Assembler	Assembler 란 J2EE 에서 정의하는 역할 중에 하나로, J2EE 어플리케이션과 J2EE 모듈을 구성하는 다양한 종류의 파일을 수집해서 조립하는 역할을 맡는다. 또한 Assembler 는 어플리케이션과 모듈의 DD 파일을 생성 또는 변경하는 일도 한다.
Authentication **인증**	Subject 의 신원을 확인하는 것으로, 본질적으로 " 당신은 누구인가?" 라는 질문에 대해 답하는 것이다.
Authorization **권한 부여**	특정 Subject 가 특정 액션을 실행할 권한이 있는지를 체크한다. 인증이 성공한 Subject 나 익명 Subject 에 대해서만 권한이 있는지를 체크한다.
Authorization outcome **권한 확인 결과**	권한이 있는지 체크하는 쿼리의 결과값. 값의 형태는 “GRANTED” 나 “DENIED” 둘 중 하나가 된다.

Term	Definition
Base 64	ASCII 문자 셋중 65 개의 문자만을 사용하여 8 진법 포맷으로 텍스트를 저장하는 방법이다. 이는 일종의 텍스트 인코딩으로 암호화와는 다르다. JEUS 보안 시스템에서 Base 64 인코딩은 패스워드를 저장하는 데 사용된다. 이는 실수로 시스템 관리자 이외의 사람이 비밀번호를 알아내는 것을 방지하고자 하는 최소한의 보호 장치이다. Base 64 로 인코딩된 비밀번호는 보안이 유지된 파일이나 데이터베이스에 저장된다.
Checked	Checked Permission 항목을 참고한다.
Checked Permission	Permission 이 특정 Principal 이나 Role 에 속해 있는 경우.
Client	"master" JVM 에 대한 클라이언트 JVM 을 일컫는 것으로, "slave" JVM 과는 다르다.
Code Subject	런타임 코드(Java 코드)에서 사용하는 특별한 종류의 Subject. Java 코드는 Code Subject 를 사용하여, 실제 호출자가 가지고 있지 않는 권한도 얻어올 수 있다. Code Subject 는 모든 Permission 을 가지고 있다.
Context id	권한 체크가 일어나는 context 를 나타내는 id. 보통 각 J2EE 모듈 (EJB, Web)은 권한 체크동안 자신만의 유일한 context id 를 가지고 있다. 각 context id 는 Authorization 시스템내에서 유일한 role-to-resource 매핑을 나타낸다.
Credential	Subject 가 소유하고 있는 보안 속성. 여러가지 목적으로 사용되지만, 일반적으로 Subject 인증시 증빙 자료로 사용된다. Credential 은 public, private(seceret) 두가지 종류가 있다. 비밀번호 (private)와 디지털 인증서 (public)가 대표적이다.

Term	Definition
Credential Factory	jeus.security.base.CredentialFactory 인터페이스를 구현하는 클래스. Subject 는 Credential Factories 를 가지고 있으며, 이는 런타임시 실제 Credential 을 생성하는데 사용된다.
Credential Mapping	사용자 이름과 Credential 매핑.
Credential Verification **Credential 유효성 검사**	제시된 Credential 이 유효하고 신뢰할만 한지 검증하는 작업 (패스워드가 정확한가?, 디지털 인증서가 유효한가?)
Deployer	Deployer 란 J2EE 에서 정의하고 있는 역할 중 하나로, J2EE 어플리케이션 및 모듈을 특정 타겟 환경에 맞게 설정하고, deploy 하는 일을 한다.
Domain **도메인**	보안 서비스 (SPI 구현 클래스)의 집합. 각 도메인은 시스템에서 유일한 이름을 가져야 한다. 도메인은 서로 다른 J2EE 어플리케이션이나 시스템이 각각 다른 보안 서비스를 사용하도록 한다.
Excluded	Excluded Permission 을 참조한다.
Excluded Permission	Excluded Permission 이 부여되면, 해당 리소스에는 누구도 접근할 수 없다. 가령, "RSC"라는 리소스에 Excluded Permission 이 부여되면, 실제 아무도 "RSC"에 접근할 수 없게 된다. Excluded Permission 은 Unchecked Permission 보다 우선 순위가 높다.
Group	동일한 이름(가령 "group1")으로 지칭되는 논리적인 Subject 의 집합. 그룹은 Subject 에서 Optional Principals(메인 Principal 과는 상반된)에 속한다. 서로 다른 Subject 들도 동일한 그룹 Principal 을 공유할 수 있다.

Term	Definition
JACC	컨테이너에 대한 Java 권한 부여 스펙(Java Authorization Contract for Containers 의 약어). 이는 커스터마이즈된 Authorization Provider 를 개발해서 J2EE 1.4 Server 에 통합하기 위한 SPI 를 정의한 스펙이다. JEUS 는 JACC 1.0 스펙을 충실히 지원한다.
JEUS_HOME	현재 JEUS 를 설치한 디렉토리
Main Principal	Subject 에 대한 유일한 id. Subject 는 단 하나의 Main Principal 을 가지고 있다. Main Principal 은 Optional Principal 과는 달리 도메인 내에서 다른 Subject 와 공유될 수 없다.
Master	보안 클러스터로 묶인 JVM 들에서 중앙 서버로 역할하는 JVM. Slave JVM 과 Client JVM 은 Master JVM 에 커넥션을 맺는다.
Non-repudiation 부인 방지	보안 관련 이벤트를 기록하기 때문에, 이후 특정 이벤트가 발생되었다는 것을 부정할 수 없게 한다. 이벤트는 암호화 되어, 로그 파일에 기록된다.
Optional Principal	Main Principal 을 제외한 모든 다른 Principal. Optional Principal 은 보통 그룹명를 나타내지만 그룹 명를 나타내지 않는 경우도 있다.
Permission	허가된 액션을 말한다. Permission 은 이름과 액션, 두가지 파트로 구성되어 있다. 가령 JNDI lookup 을 실행하는 Permission 은 이름은 "jeus.server.jndi"이고 액션은 "lookup"으로 구성되어 있다. 모든 Permission 은 java.security.Permission 인터페이스를 구현한다.

Term	Definition
Principal	액션을 수행하는 실체의 유일한 id. 실례로 사용자 명이나 그룹명이 해당 된다.
Private Credential	민감한 정보를 다루기 때문에 보안이 필요한 Credential 이다. 예로 패스워드가 Private Credential 이다. "Credential" 항목을 참고하기 바란다.
Public Credential	민감한 정보를 다루지 않기 때문에 보안이 필요하지 않는 Credential. 예로 디지털 인증서가 Public Credential 이다. "Credential"항목을 참고하기 바란다.
Resource	Principal 에 의해서 액션이 적용될 수 있는 자원들의 집합이다.
Role	Principal 의 논리적인 그룹이다. Role 은 보통 Principal 의 집합이다 (하나의 Main Principal 과 여러 개의 Group Principals 로 구성되어 있다). J2EE 어플리케이션 성격에 따라 각각 다른 Role 이 적용된다. 어플리케이션의 Role 을 deploy 할 환경에 미리 설정되어 있는 실제 Principal 에 매핑하는 것은 deployer 의 역할이다 (deployer 는 실제 Principal 들이 적합한 Role Permission 을 갖도록 설정한다).
Security Domain **보안 도메인**	"도메인" 항목을 참조한다.
Security Service **보안 서비스**	"서비스" 항목을 참조한다.

Term	Definition
Security SPI **보안 SPI**	"SPI" 항목을 참조한다.
Security System **보안 시스템**	보안 관련 코드, jeus.security.* 패키지의 모든 클래스, 보안 클래스가 사용하는 보안 데이터 저장소, 외부 관련 설정 파일등, 보안과 관련된 모든 것을 지칭하는 포괄적인 용어다. 보안 시스템은 이 문서에서 언급하고 있는 모든 것이라 해도 과언이 아니다.
Service **서비스**	특정 보안 기능을 제공하기 위해서, 해당 SPI 클래스 를 규약에 따라 구현한 Java 클래스. 예로 authentication 서비스와 authorization service 가 있다. 이 용어는 보안 서비스 프레임워크의 최상위 레벨에 있는 추상 클래스를 지칭하기도 한다.
SHARED_DOMAIN	특별한 도메인으로 여기서 지정한 보안 서비스는 모든 다른 도메인과 공유된다. 따라서, SHARED_DOMAIN 은 모든 다른 도메인에서 공통적으로 적용되는 보안 서비스를 포함하고 있어야 한다.
Slave	"master" JVM 에 커넥션을 맺는 JVM 이나, "client" JVM 과는 다르다. "Slave" JVM 은 일종의 이차 보안 서버라 할 수 있으며, 보안 클러스터를 구성하는 멤버이다.

Term	Definition
SPI	서비스 제공 인터페이스(Service Provider Interface). 구현되어야 할 추상 메소드를 포함하고 있는 추상 클래스이다. 모든 SPI 는 자신만의 보안 서비스를 구현하도록 하고 있다. JEUS 에서는 jeus.security.spi 패키지에 구현 클래스가 포함되어 있다. 각각의 SPI 클래스는 서로 다른 보안 기능을 가지고 있는데, 가령 인증 관련 SPI, 권한 부여 관련 SPI 가 있다. SPI 의 구현 클래스를 서비스라고 한다.
Subject	액션을 실행하는 실체. Subject 는 구체적인 사람일 수도 런타임 코드일 수도 있다. 각 Subject 는 정확히 로그인시 하나의 도메인에 할당된다. Subject 는 유일한 Main Principal 을 갖고 있으며, 사용자 그룹을 나타내는 다양한 부가적인 Principal 을 가지고 있다. Subject 는 Credential 과 CredentialFactory 를 가지고 있다.
Subject Factory	Subject 를 생성하는 보안 서비스로, Subject 는 Principal 과 Credential 을 가지고 있다. Subject Factory 는 Subject 생성을 보안 시스템에 위임할 때 사용된다.
Subject Validation **Subject 유효성 검증**	해당 Subject 가 유효한지 체크하는 보안 서비스. 유효한 Subject 는 "blocking/denying" Credential 을 가지고 있지 않다 (Credential 에 락이 걸려 있다던지, Credential 의 만료일자가 지났을 경우, "blocking/denying" Credential 이라고 한다).
SYSTEM_DOMAIN	J2EE 어플리케이션과 모듈에 사용하는 보안 도메인 이름. 도메인 명은 JEUSMain.xml 에 명시적으로 설정하지 않는다.

Term	Definition
SYSTEM_DOMAIN	표준 관리 툴을 사용해 JEUS Server 를 관리할 때 적용되는 도메인 명이다. 이 도메인은 JEUS administrator 계정과 "boot"와 "down" 같은 서버 관리 Permission 들을 포함하고 있다.
Unchecked	Unchecked Permission 을 참조한다.
Unchecked Permission	모든 사람이 소유하는 Permission. 가령 "RSC" 리소스에 Unchecked Permission 이 부여되어 있다면, 누구나 "RSC"리소스에 접근할 수 있게 된다. Unchecked permission 은 Excluded Permission 보다 우선 순위가 낮으나, Checked Permission 보다는 우선 순위가 높다.
Unspecified EJB method	ejb-jar.xml 파일의 <method permission> 태그에 포함되어 있지 않는 EJB 메소드. JEUS 는 이 메소드를 특정 Role 에 매핑하거나, Excluded 또는 Unchecked (디폴트)로 취급한다. 웹 모듈에서, web.xml 에 특별한 보안 설정이 없는 Servlet URL 일 경우, 항상 Unchecked 로 취급한다 (Unchecked 라는 말은 권한 체크 과정을 거치지 않기 때문에, 누구나 자유롭게 접근할 수 있다는 것을 뜻한다).

연락처

Korea
Tmax Soft Co., Ltd
18F Glass Tower, 946-1, Daechi-Dong, Kangnam-Gu
Seoul 135-708
South Korea
Email: info@tmax.co.kr
Web (Korean):http://www.tmax.co.kr

USA
Tmax Soft, Co.Ltd.
2550 North First Street, Suite 110
San Jose, CA 95131
USA
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com

Japan
Tmax Soft Japan Co., Ltd.
6-7 Sanbancho, Chiyoda-ku,
Tokyo 102-0075
Japan
Email: info@tmaxsoft.co.jp
Web (Japanese): http://www.tmaxsoft.co.jp

China
Beijing Silver Tower, RM 1507, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
Tel: 86-10-64106148 Fax: 86-10-64106144
E-mail : info@tmaxchina.com.cn
Web (Chinese):http://www.tmaxchina.com.cn

1 소개

일상 생활에서 security 이라는 개념은, 중요한 무엇인가를 외부로부터, 혹은 내부로부터의 침입이나 훼손 등의 우려가 있을 때에 이를 차단하고 예방하기 위해서 사용된다. 이러한 security 의 개념은 컴퓨터를 사용하거나 엔터프라이즈 환경에서 사용자들의 정보를 다루고 서비스를 제공하는 데에도 유사하게 적용될 수 있다.

이 때에는 security 의 대상이 되는 것은 사용자들의 이름과 비밀 번호, 주소 등의 개인적인 정보와 함께 서비스를 제공하거나 시스템을 동작시키는 리소스로 볼 수 있으며, 각각의 시스템에서는 중요한 정보와 시스템의 리소스를 보호하기 위해서 보안 관련 소프트웨어를 사용하거나 하드웨어적인 보안 시스템을 구축하게 된다.

JEUS 에서는 JEUS 에 등록된 리소스와 사용자의 정보를 보호하기 위해서 다양한 security 서비스를 제공한다.

JEUS Security 안내서는 JEUS security 구조와 보안 정보 관리와 관련된 일반적인 정보에 대해 자세히 설명하고 있다.

2 보안 시스템의 개요

2.1 소개

본 장에서 보안 시스템에 대해 개괄적으로 설명하겠다. 본 장은 다시 다음과 같은 세부 절로 나뉜다.

1. 보안 시스템 설계의 목표
2. 보안 시스템의 주요 특징
3. 보안 시스템 구조
4. 보안 시스템의 주요 개념

2.2 설계 목표

JEUS 보안 시스템은 다음의 목표를 두고 설계 되었다.

- 유연하고 교체하기 쉬운 프레임워크: 기존의 3rd-party 보안 메커니즘을 장착한 보안 시스템과 효율적으로 통합되고, 다양한 데이터 스토리지를 지원해야 한다.
- 기밀성 유지: 뛰어난 해커가 좋지 않은 의도를 가지고 보안 시스템 소스를 디컴파일 한다 하더라도, 권한이 없는 시스템에는 결코 접근할 수 없어야 한다. 이 목표는 상당 수준 충족되어 있으나, 완벽한 보안 시스템은 있을 수 없다는 사실을 상기하기 바란다.
- 성능에 무리가 없어야 한다: 시스템과 어플리케이션 코드에 보안 시스템이 관여 하더라도 전체 퍼포먼스에는 최소한의 영향만 미쳐야 한다. 일반적으로 보안이 강화될수록, 성능이 떨어 지는 경향이 있다. 최대한의 보안 기능을 제공하면서 가능한한 빨리 처리할 수 있어야 한다.
- 유지 보수를 쉽게 하고 3rd party 보안 서비스를 쉽게 생성하기 위해, 간결하고 명료한 API 와 SPI 가 제공되어야 한다.

- 유지 보수를 쉽게 하고 3rd party 보안 서비스를 쉽게 생성하기 위해, 문서화가 잘 되어 있어야 한다.
- 무결성: 아주 작은 버그만이 허용되어야 한다.
- 표준에 따라야 한다.

JEUS 보안 시스템은 위에서 언급한 설계상의 목적을 상당히 충족하고 있다. 이 문서를 읽고, 실제 보안 시스템을 작동시켜 보면, 아마 여러분도 동일한 결론에 도달할 것이다.

2.3 주요 특징

JEUS 보안 시스템의 주요 특징들은 다음과 같다.

- 공개된 아키텍처와 유연한 프레임워크를 가지고 있어, 기존에 사용중이던 3rd party 보안 시스템과 효과적으로 통합할 수 있다.
- 적절하게 설정되었다면, 좋지않은 의도를 가진 타인에 의한 악의적인 공격으로부터 시스템을 효과적으로 보호할 수 있다. 그러나, 완벽한 보안 시스템이란 없으므로, 방화벽과 같은 추가적인 보안 메커니즘을 도입하기 바란다.
- 인증에 사용되는 credential 정보와 보안체제는 패스워드 개념 이상의 개념을 가지고 있다.
- 소위 동적인 principal-to-role, role-to-resource 매핑을 지원한다. 동적인 매핑이란 실제 권한 부여 매핑(principal-to-role, role-to-resource 매핑)이 런타임시에 계산된다는 것을 뜻한다. 이는 다음과 같은 보안 정책을 가능하게 한다. "사용자 U 는 월요일부터 금요일까지 업무 시간인 9 시부터 5 시까지 R 이란 Role 을 부여받는다", "모든 사람이 R 이란 Role 을 부여받는다" 라던가 "아무도 R 이란 Role 을 부여 받을 수 없다"
- "보안 도메인" (단순히 도메인이라고 표현한다) 개념을 지원한다. 이는 서로 다른 J2EE 어플리케이션들이 각각의 특성에 맞는 별도의 보안 서비스를 이용할 수 있게 한다.
- 인증, 권한 부여, Repositories, 감사, 클러스터링등과 같은 중요한 보안 기능에 대한 디폴트 구현을 제공한다.

- 유연한 이벤트 핸들링 모델을 통한 보안 감사 메커니즘을 지원한다.
- Servlet, EJB, Application 소스코드 내에 직접 보안 기능을 추가할 수 있다. 이는 Subject 와 Policy 를 추가하는 것과 같은 단순한 보안 기능에 한해 가능하다.
- J2EE 1.4 와 JACC 1.0 스펙을 완벽히 지원한다.
- 본 매뉴얼과 추가적인 Javadoc 을 통해 문서화를 충분히 지원하고 있다.
- 다른 JEUS 모듈과 독립적이다. 따라서, JEUS 시스템에서 보안 시스템만 따로 떼어내어 다른 컨텍스트에 적용하기가 상대적으로 쉽다.

2.4 시스템 구조

보안 시스템의 기본 아키텍처는 다음 [그림 1]에 요약되어 있다.

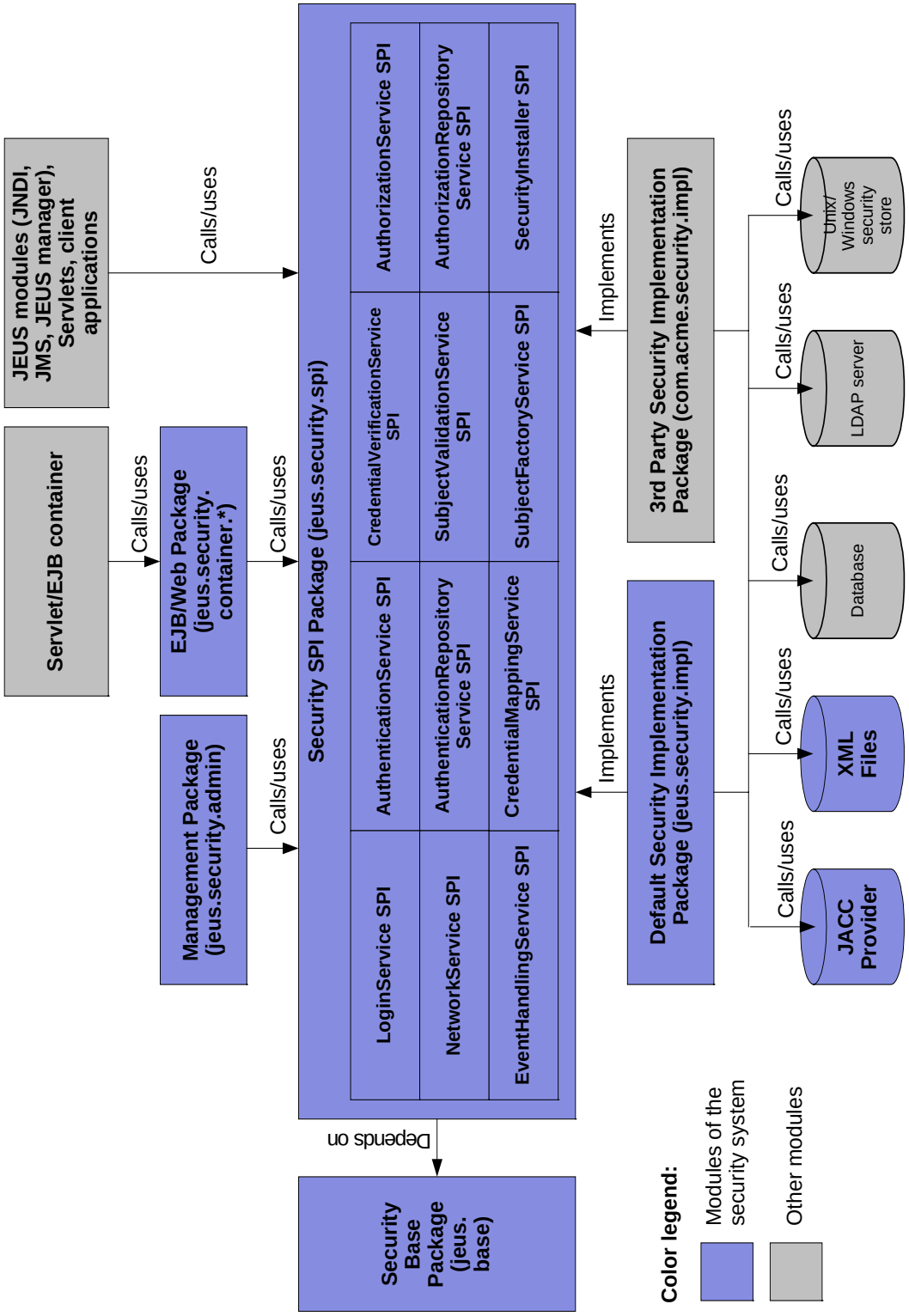
JEUS modules (JNDI, JMS, JEUS manager), Servlets, client applications
Calls/uses
Servlet/EJB container
Calls/uses
EJB/Web Package (jeus.security.container.*)
Calls/uses
Management Package (jeus.security.admin)
Calls/uses
Security SPI Package (jeus.security.spi)
LoginService SPI
NetworkService SPI
EventHandlingService SPI
AuthenticationService SPI
AuthenticationRepository Service SPI
CredentialMappingService SPI
CredentialVerificationService SPI
SubjectValidationService SPI
SubjectFactoryService SPI
AuthorizationService SPI
AuthorizationRepository Service SPI
SecurityInstaller SPI
Depends on
Security Base Package (jeus.base)
Implements
3rd Party Security Implementation Package (com.acme.security.impl)
Implements
Default Security Implementation Package (jeus.security.impl)
Calls/uses
Unix/ Windows security store
Calls/uses
LDAP server
Calls/uses
Database
Calls/uses
XML Files
Calls/uses
JACC Provider
Color legend:
Modules of the security system
Other modules

그림 1 보안 시스템 구조

위의 그림에서 주요 컴포넌트는 다음과 같다:

- Package **jeus.security.spi**: 보안 시스템의 가장 핵심 부분인 Security SPI 클래스로, 중앙에 큰 박스로 구분되어 있다. 이 추상 클래스들에는 third party 제품에서 구현해야 하는 추상 메소드 뿐 아니라, JEUS 컨테이너를 포함한 사용자 코드에서 호출하는 메소드들도 포함되어 있다. 보다시피, 현재 12 개의 사용가능한 SPI 클래스가 있으며, 각각은 인증, 권한 부여와 같은 보안의 핵심적인 기능을 담당하고 있다.

- Package **jeus.security.base**: 왼쪽의 작은 박스로 구분되어 있으며, Security SPI 가 참조하는 인터페이스와 구현 클래스가 포함되어 있다. 이 패키지에서 중요한 두가지 클래스는 Subject 와 Policy 이다.

- Package **jeus.security.impl.***: 아래쪽 가운데 박스로 구분되어 있으며, JEUS 에서 제공하는 SPI 클래스의 디폴트 구현 클래스가 포함되어 있다. 디폴트 구현 클래스는 현재 XML 파일 Repository 와 JACC Provider 를 지원하고 있다.

- Packages **jeus.security.admin**, **jeus.security.container**: 위쪽 가운데 있는 박스로 구분되어 있으며, SPI 클래스를 호출하는 코드를 포함하고 있다.

- 그림의 가장 하단에 있는 것은 Repository 와 외부 보안 메커니즘을 나타내고 있다.

  Repository 로는 데이터베이스, LDAP 서버, XML 파일이 있다. Repository 는 Subject 와 Policy 와 같은 보안 요소들을 영구적으로 저장하는데 사용된다.

  외부 보안 메커니즘이란 인증 또는 권한 부여가 실행되는 메커니즘을 말한다. 외부 보안 메커니즘의 대표적인 예는 JACC Provider 이다. 일반적으로 Repository 와 보안 메커니즘 사이의 경계를 명확히 긋기 힘들다. SPI 구현 클래스(여기서는 jeus.security.impl.* 패키지에 포함된 클래스)가 실제 어떤 Repository 와 보안 메커니즘을 적용할 지 결정한다.

2.5 주요 개념

보안 시스템 아키텍쳐와 관련된 몇 가지 중요한 개념에 대해 알아보도록 하자. 이는 보안 시스템을 이해하는데 필수적이다.

여기서는 특별히 다음의 중요 개념에 대해서 설명한다.

- 로그인
- 인증
- 권한 부여
- 감사
- 서비스와 SPI
- 도메인

2.5.1 로그인 개념

JEUS 보안 시스템에서, 로그인이란 Subject 를 실행 쓰레드(Java 쓰레드)에 결합시키는 것을 말한다. 이 개념은 인증과는 다르다. 일반적으로 Subject 가 실행 쓰레드와 결합되기 전에 인증이 일어난다. 만약 성공적으로 인증되면 로그인은 정상적으로 진행되어, Subject 와 쓰레드의 결합하게 되지만, 인증에 실패하면 로그인은 더 이상 진행되지 않는다.

Subject 가 성공적으로 로그인한 후에, 로그인된 Subject 를 대상으로 권한 체크가 이루어 진다. 따라서, 로그인은 인증과 권한 체크, 양쪽 모두에 걸쳐있다고 할 수 있다. 로그인의 반대 개념은 로그아웃으로, 현재 Subject 를 실행 쓰레드로부터 분리하는 과정이다.

로그인 이면에는 다음과 같은 스택 기반 오퍼레이션이 있다. 여러 개의 서로 다른 Subject 로 로그인했을 경우, 최근에 로그인한 Subject 부터 스택의 최상단에 차곡차곡 쌓이게 된다. 그리고 나서, 권한 체크에는 스택의 최상단에 놓인 Subject(가장 최근에 로그인한 Subject)를 사용한다. 이 Subject 가 로그아웃 하고 나면, 스택으로부터 제거되고, 바로 직전에 로그인한 Subject 가 노출되어 활성화 된다. 이 메커니즘을 [그림 2]에서 보여주고 있다.

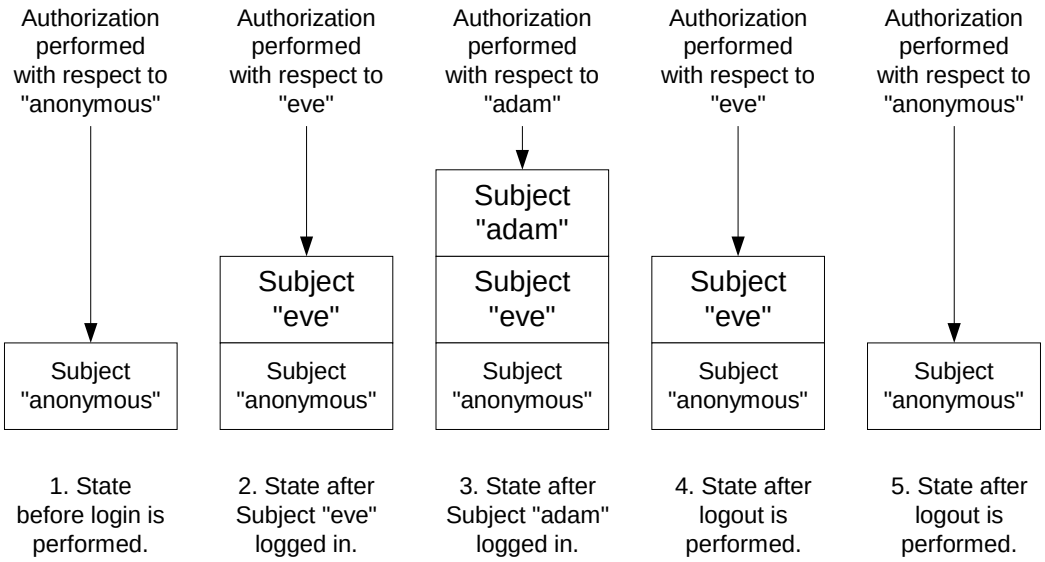

그림 2 스택 기반의 로그인 메커니즘

이전에 보안 아키텍쳐 그림에서도 보았다시피, 로그인 메커니즘은 `jeus.security.spi.LoginService SPI` 에 의해 구현된다. 이 클래스에 대한 자세한 정보는 9 장을 참고 하기 바란다.

JEUS 보안 시스템에서, Java 쓰레드당 하나의 로그인 스택을 가질 수 있다.

2.5.2 인증 개념

인증이란 간단히 말해서 호출자의 신원을 파악하는 작업인데, 이 후 권한 체크시 호출자의 신원을 사용하기 위해서다.

JEUS 보안 시스템에서, 신원이란 java.security.Principal 인터페이스로 정의한 Principal 을 말한다. Subject 는 이러한 Principal 들을 속성값으로 저장하고 있다. `Subject` 란 `jeus.security.base.Subject` 클래스를 말하며, 이전 절에서 언급한 로그인시 실행 쓰레드와 결합하는 주체이기도 하다. `Subject` 는 항상 `Principals` 과 `Credentials` 을 보안 속성으로 포함하고 있다.

주의: JEUS 보안 시스템에서 말하는 Subject 는 `javax.security.auth.Subject` 클래스가 정의하는 JAAS 의 `Subject` 와는 다른 것이다. 그러나, 이 두 클래스는 많은 공통점이 있어, 한쪽에서 다른 쪽으로 전환하는 것이 가능하고, 일부 정보를 잃더라도 `alias` 로 지칭될 수 있다.

JEUS Subject 는 UML 로 다음과 같이 표기할 수 있다 [그림 3].

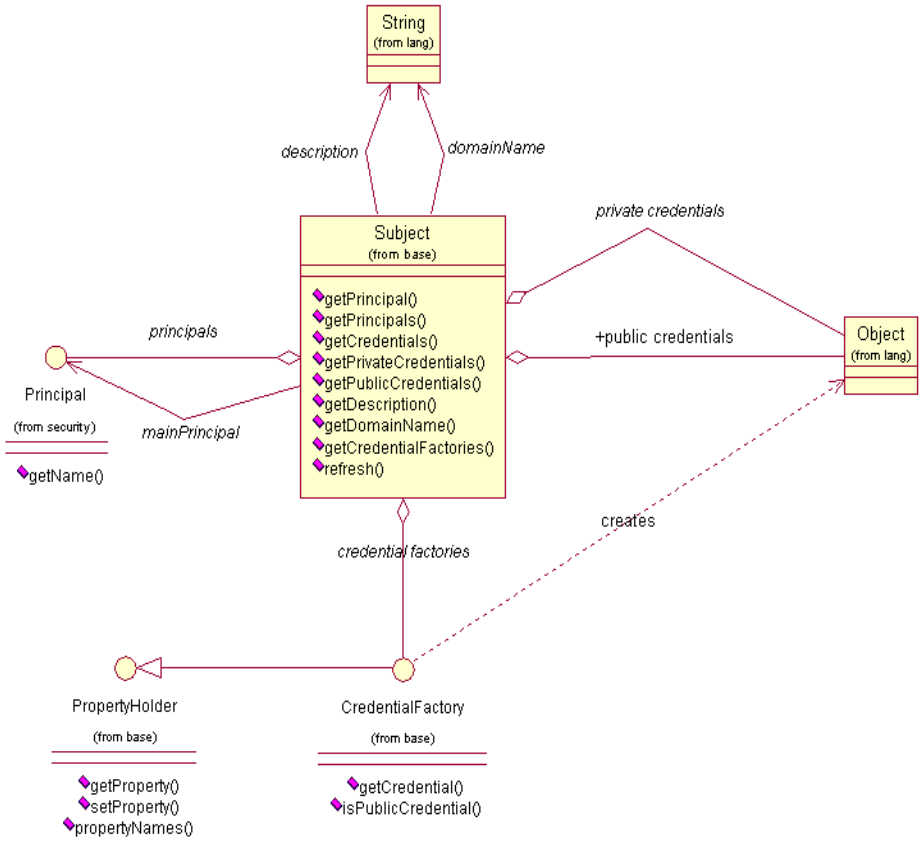

그림 3 Subject UML 다이어그램

위에서 보다시피, Subject 는 유일한 id 인 메인 Principal 을 가지고 있다. 또한 메인 Principal 외의 여러 개의 부가적인 Principal 을 가질 수 있는데, 이러한 부가적인 Principal 들은 다른 Subject 와 공유할 수 있다(따라서, 부가적인 Principal 을 그룹 Principal 이라고도 한다).

Subject 는 또한 public 또는 private Credentials 을 가질 수 있다. Credential 이란 보통 Subject 의 신원을 확인하거나, 특정 정보를 전달하려는 목적으로 사용되는 증빙 자료이다. 대표적인 private Credential 로는 패스워드가 있고, public Credential 로는 디지털 인증서를 꼽을 수 있다.

Subject 는 소위 CredentialFactory 를 사용하여 Credential 들을 생성한다. Subject 의 refresh()라는 메소드가 실제 CredentialFactory 로부터 Credential 을 얻어 와서, 이를 public 또는 private Credential set 에 추가한다.

Subject 는 반드시 하나의 도메인에 속해야 한다는 것을 명시해야 한다. 도메인 A 에 속하는 Principal "peter"는 도메인 B 에 속하는 Principal "peter"와는 별개의 것이다.

2.5.3 권한 체크(부여) 개념

JEUS 보안 시스템에서, 권한 체크란 이미 성공적으로 인증된 Subject 가 특정 액션을 실행시킬 권한이 있는지 없는지 확인하는 것을 말한다.

권한 체크는 보통 시스템 레벨에서 일어난다. 특정 Subject(보통 administrators)가 JEUS Server 를 부트 또는 다운시킬 권한을 가지고 있는지 없는지 체크하는 것이 여기에 해당한다. 또한 권한 체크는 어플리케이션 레벨에서도 일어난다. 원격지 호출자가 특정 어플리케이션 컴포넌트 (특정 EJB 메소드의 실행, 특정 Servlet 호출)에 접근할 수 있는지 없는지 JEUS 엔진에서 확인하는 것은 여기에 해당한다.

J2EE 에서와 마찬가지로, JEUS 보안 시스템에서 권한 부여는 Role-Based 메커니즘이다. 즉, J2EE 어플리케이션 Assembler 또는 개발자가 Role 에 따라 보안 설정(security restriction)을 해두면, Deployer 또는 시스템 관리자가 어플리케이션을 deploy 하면서 실제 시스템의 Principal 을 논리적인 Role 에 매핑한다. Role-Based 접근방법은 J2EE 어플리케이션에서뿐 아니라, JEUS 시스템과 관련된 권한 체크에도 사용된다는 것을 명심하기 바란다.

더 나아가 JEUS 보안 시스템에서 개개의 권한 매핑을 Permission(실제 `java.security.Permission` 의 서브 클래스)이라 한다. 가령, `Principal "peter"`는 `"R"`이라 불리는 `Role` 에 접근할 수 있는 `Role Permission` 을 가지고 있다고 하자(이를 `"peter"`는 `Role "R"` 에 포함되어 있다고 한다). 그리고, `Role "R"`은 `"jndi"`리소스에 접근해 `"lookup"`액션을 실행할 수 있는 `Resource Permission` 을 가지고 있다고 하자. 이러한 개념은 [그림 4]에 나타나 있다.

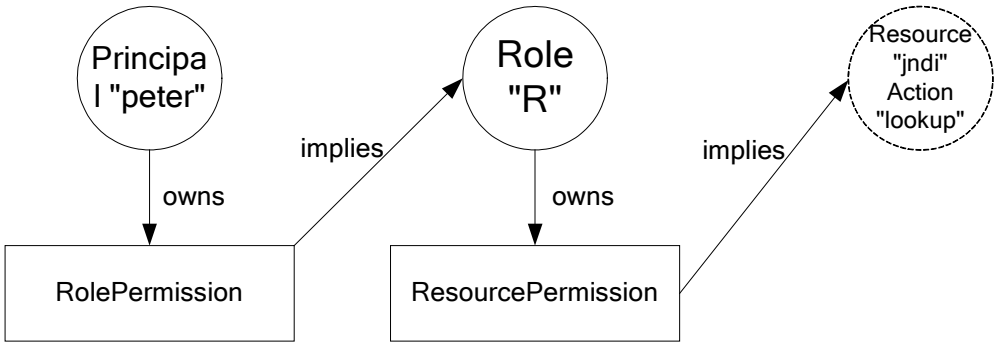

그림 4 Role 기반 Permission 체크

위 그림에서 보듯이, Principal 과 Role 만이 실제 물리적 실체로서 모델링 된다. 그림에서 리소스를 감싸고 있는 원은 점선으로 표시되어 있는데, 리소스는 실제 권한 부여 시스템에서 물리적으로 모델링되는 부분이 아닌 암시되는 부분이기 때문이다.

위 그림에서, “RolePermission” 과 “ResourcePermission”은 실제로는 각각 jeus.security.resource.RolePermission 클래스와 jeus.security.resource.ResourcePermission 클래스의 인스턴스를 나타낸다. 이 두 클래스는 java.security.Permission 추상 클래스로부터 확장된 대표적인 클래스들로, 이들 외에도 다양한 Permission 클래스의 서브 클래스들이 있다.

다시 한 번 정리하면, Principal 은 여러 개의 Role Permission 을 가지고 있고, 각 Role Permission 은 특정 Role 이 Principal 을 포함하는 것을 허용한다. 다시 Role 도 여러 개의 Resource Permission 을 가지고 있는데, 각 Resource Permission 은 해당 Role 이 특정 리소스들에 대해 특정 액션들을 취할 수 있도록 허락한다. 따라서. Principal-Role-Resource 를 연결시켜서 생각해 보면, 어떤 Principal 이 특정 리소스에 대한 특정 액션을 실행할 수 있을지 없을지 판단할 수 있다.

위의 그림에서, Principal "peter"는 실선으로 RolePermission 을 소유하고 있다("owns")라고 표시되어 있다. 그러나, RolePermission 은 대각선으로 Role "R"을 암시하고 있다("implies")고 표시되어 있다. "암시하다" 라는 말은 둘 사이의 관계가 정적이지 않다는 뜻으로, 때때로 매핑될 수도 있고, 그렇지 않을 수도 있음을 말한다. 즉, 런타임에 RolePermission 이 해당 Role 을 암시할지 안 할지는 동적으로 결정된다는 말이다.

실제로 RolePermission 의 implies(Permission p) 메소드에 의해 결정되는데, implies(..)메소드가 true 를 리턴하면, RolePermission 이 Role "R"을 암시하지만, "false"를 리턴하면, 암시하지 않는다. 따라서, 전자의 경우에는 Principal "peter"가 Role "R"에 포함되지만, 후자의 경우에는 포함되지 않는다. 같은 논리가 Role "R"과 리소스 "jndi"사이에도 성립된다.

Permission 이란 용어에 대한 좀 더 자세한 내용은 J2SE Javadoc 에서 java.security.Permission 부분을 참고하기 바란다.

위의 정보를 토대로, 동적인 매핑을 구현하기란 실제 어려운 일이 아니다. 단지 implies(..) 메소드의 구현을 변경하면, 특정 조건하에서만 Permission 이 Role 또는 리소스를 암시하도록 만들 수 있다. 가령, 동적인 매핑을 이용해, Principal "peter"가 업무 시간(9 시부터 5 시) 동안만 Role "R"에 포함되게 구현할 수 있다. 이는 새로운 RolePermission 서브클래스 (이하 `TimeConstrainedRolePermisson`)를 생성하고, implies(..) 메소드 내에서 시간 제약과

관련된 코드만 작성해 놓으면 된다. 예로 [그림 5]와 [그림 6]을 보기 바란다.

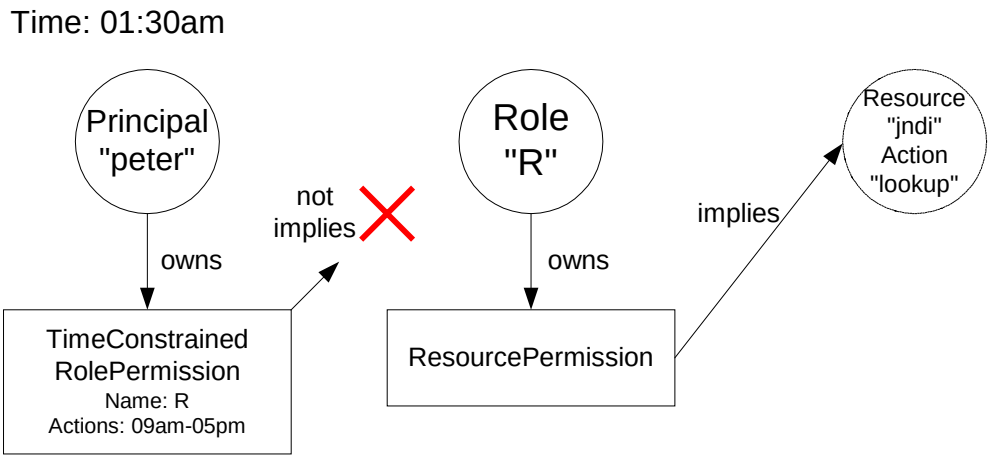

그림 5 오전 1 시 30 분의 Role

[그림 5] 에서 보듯이 오전 1 시 30 분에는 peter 는 Role R 의 권한이 없다.

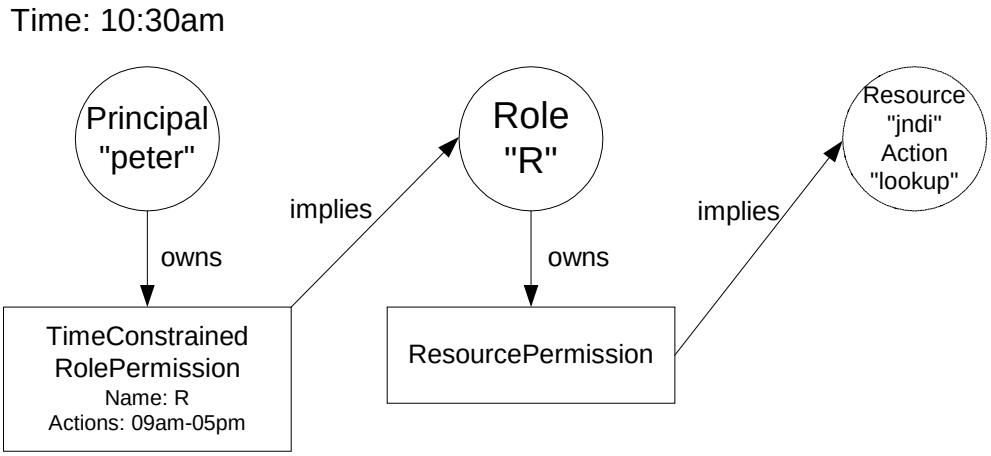

그림 6 오전 10 시 30 분의 Role

위 그림에서 보다시피, `TimeConstrainedRolePermission` 두가지 값인 Name 과 Actions 를 가지고 있다. Name 은 타깃 Role 의 이름인 "R"이고, Actions 는 유효한 시간 (9 시부터 5 시까지)을 나타내고 있다. "Name"과 "Actions"는 대부분의 `java.security.Permission` 구현 클래스의 변수들로 선언된다.

기본적인 Permission-Based 매핑과는 별도로, 권한 부여 시스템에서 모든 Permission 은 다음 3 가지 카테고리 중에 하나에 속한다.

1. **Excluded permission**: 누구도 해당 Permission 에 대한 허가를 가질 수 없다. 예로 리소스 "RSC"에 대한 접근을 Excluded Permission 으로 설정했다면, 아무도 "RSC"리소스에 접근할 수 없게 된다. Excluded Permission 은 Unchecked Permission 보다 우선 순위가 높다.

2. **Unchecked permission**: 누구나 해당 Permission 에 대한 허가를 가질 수 있다. 예로 리소스 "RSC"에 대해 Unchecked Permission 이 설정되어 있다면, Role 에 상관없이 누구나 "RSC" 리소스에 접근할 수 있게 된다. Unchecked Permission 은 Excluded Permission 보다 낮은 우선순위를 가지나, Checked Permission 보다 높은 우선순위를 가진다.

3. **Checked permission**: 특정 Principal 이나 Role 에만 허가된 Permission 으로, 지금까지 계속 설명한 Permission 이 여기에 해당한다.

이에 대한 몇 가지 예제들은 다음에 더 살펴 보기로 하겠다.

JEUS 보안 시스템에서 모든 Permission 매핑들은 jeus.security.base.PermissionMap 클래스를 구현한 클래스이다. PermissionMap 클래스는 다시 jeus.security.base.Policy 클래스에 포함된다.

Policy 는 두 가지 종류의 PermissionMap 을 가지고 있다. pincipal-to-role map ("Role Policy" 라고 한다), role-to-resouce map("Resource Policy" 라고 한다). 이들은 각각 principal-to-role 매핑과 role-to-resource 매핑을 나타낸다. 각 PermissionMap 은 다시 위에서 언급한 3 가지 종류의 Permission 들을 (Excluded, Unchecked, Ckecked Permissions) 포함하고 있다. Checked Permission 의 경우, Role PermissionMap 과 Permission 은 Principal 이나 Role 을 매개로 결합되어 있고, Policy 와 Resource PermissionMap 은 context id (권한 체크가 일어나는 범위를 나타낸다)를 통해 결합되어 있다[그림 7].

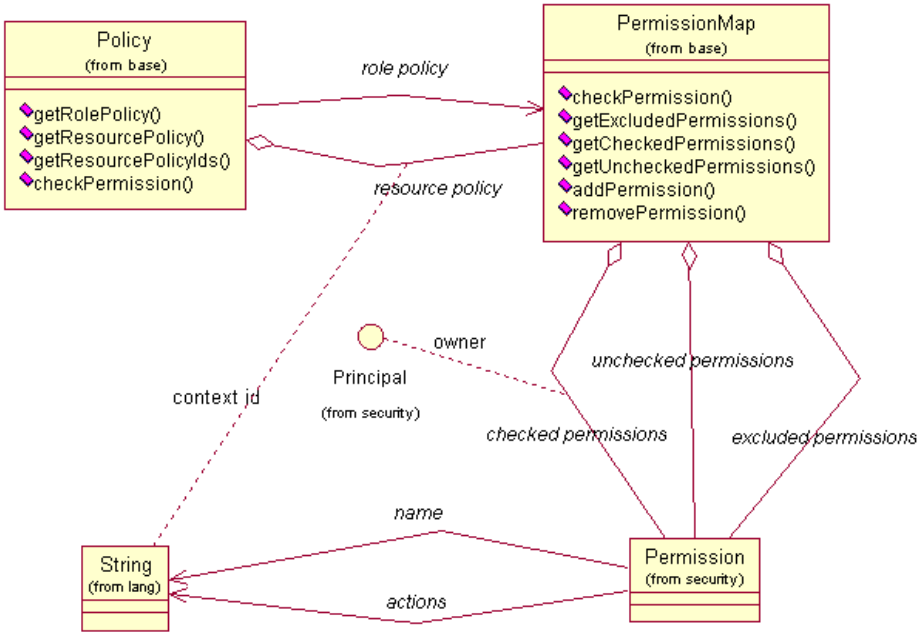

그림 7 Policy 와 PermissionMap 의 UML 다이어그램

이상에서 언급한 내용이 다음 그림에 요약되어 있다.

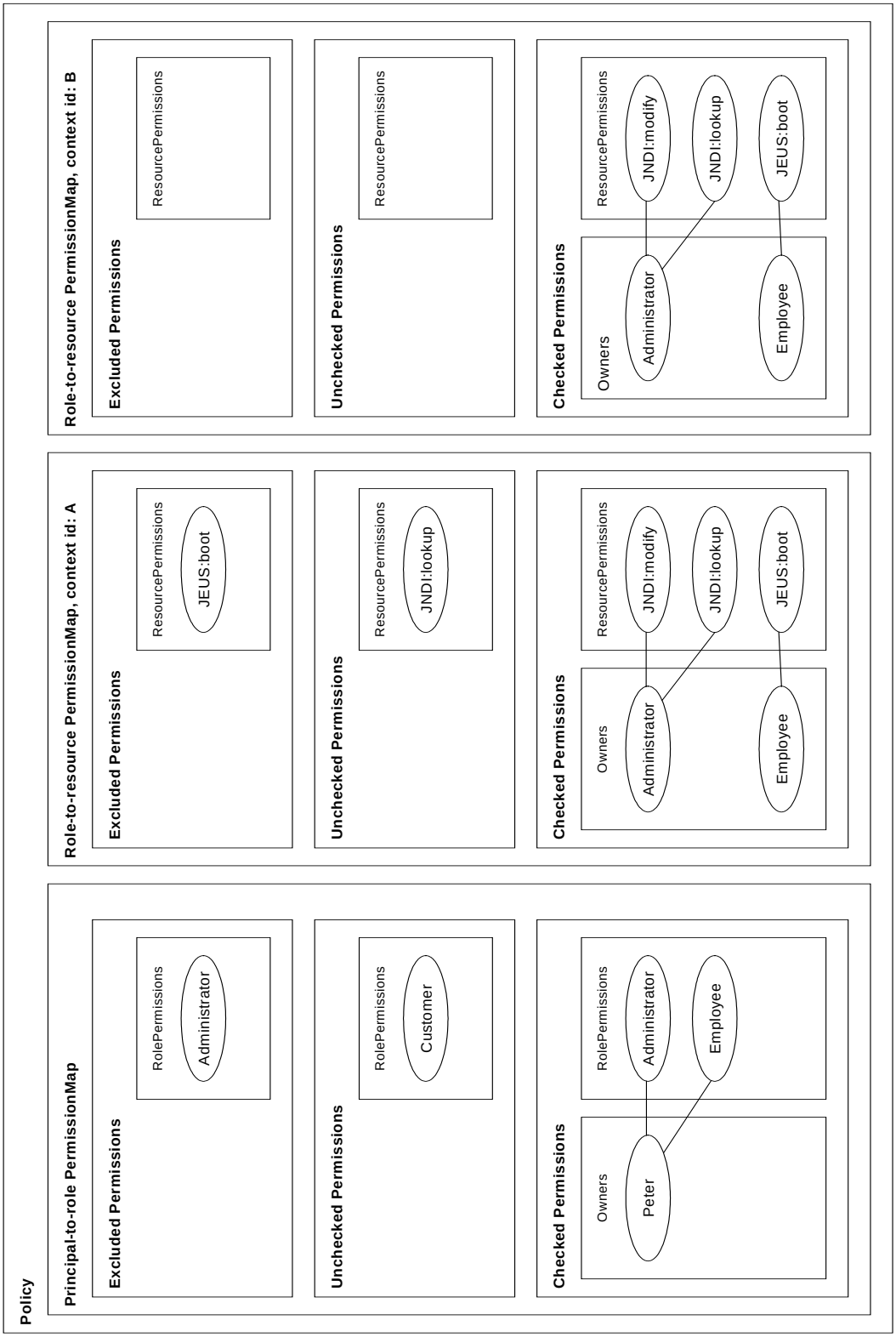

그림 8 하나의 principal-role map 과 두개의 role-resource map 을 가진 Policy 의 예제

위의 그림에서 보면, 해당 Policy 는 하나의 principal-to-role map(필수 사항)과 각각의 context id 가 "A" 와 "B"인 두개의 role-to-resource map 을 포함하고 있다. 세가지 PermissionMap 은 다시 세가지 Permission set 을 가지고 있다. Excluded, Unchecked, Checked permissions 가 그것이다. 박스내의 타원은 Name 과 Action 을 가진 실제 Permission 인스턴스(Name: Action)를 나타내고 있다.

자. 이제 다음 상황을 가정해 보자. context id 가 "A"이고, Principal 이 "Peter"인 Subject 가 "JNDI"에 접근해서 "modify"라는 액션을 실행하고 싶다고 하자. 과연 "Peter"는 원하는 액션을 실행할 수 있을까?

권한 부여 시스템 은 이 요구사항을 다음과 같은 방식으로 풀어 나간다.

1. 이름이 "JNDI"이고 액션이 "modify"인 새로운 Resource Permission 을 생성한다 (RSP 라고 부르도록 하자).

2. RSP 는 context id "A"와 Principal "Peter"와 함께 Policy 에 넘겨진다.

3. context id "A"에 대한 role-to-resource PermissionMap 이 Policy 에서 선택된다.

4. Policy 는 PermissionMap "A"의 Excluded Permission 에 RSP 가 포함되는지 체크한다. 체크 결과, RSP 는 Excluded Permission 이 아니다.

5. Policy 는 PermissionMap "A"의 Unchecked Permission 에 RSP 가 포함되는지 체크한다. 체크 결과, RSP 는 Unchecked Permission 이 아니다.

6. Policy 는 PermissionMap "A"의 Checked Permission 에 RSP 가 포함되는지 체크한다. 체크 결과 RSP 는 Checked Permission 중에 하나이다.

7. Policy 는 RSP 를 암시하는 Permission 의 소유자를 알아낸다: "Administrator"라 불리는 Role 이 바로 소유주다.

8. Policy 는 "Administrator"라는 이름을 가진 RolePermission 을 생성한다. 이를 "RLP"라고 부르도록 하자.

9. Policy 는 principal-to-role PermissionMap 을 꺼내온다.

10. Policy 는 principal-to-role Permission 의 Excluded Permission 에 포함되는지 체크한다. 체크 결과, RLP 는 Excluded Permission 이다.

11. RLP 가 excluded permission 에 포함되어 있기 때문에, Policy 는 "Administrator"라는 Role 이 Excluded Permission 이므로, 누구도 접근할 수 없다는 결론을 내릴 수 있다. 따라서, 전체 권한 체크 과정은 종결되고, "DENIED"라는 결과가 리턴된다.

위에서 보듯이, 비록 "Peter"가 "Administrator" Role 에 매핑되어 있지만, "Administrator" Role Permission 이 Excluded Permission 이므로, "Peter"를 포함한 누구도 "Administrator" Role 에 접근할 수 없게 된다. 이는 Excluded Permission 이 Unchecked Permission 과 Checked Permission 보다 우선 순위가 높기 때문이다.

위에서 설명한 대로 차근 차근 과정을 따라가 보면, 다음 권한 체크 질의가 어떻게 풀릴지 알 수 있을 것이다[표 1].

표 1 다양한 권한 체크 질의와 결과에 대한 예제

Principal	**Operation**	**Context**	**Outcome**
Peter	JNDI:lookup	A	GRANTED
Peter	JNDI:modify	A	DENIED
Peter	JEUS:boot	A	DENIED
Peter	JEUS:boot	B	GRANTED
Anonymous	JNDI:lookup	A	GRANTED
Anonymous	JNDI:lookup	B	DENIED
Anonymous	JEUS:boot	A	DENIED
Anonymous	JEUS:boot	B	DENIED

2.5.4 보안 감사의 개념

보안 감사란 일반적으로 보안과 관련된 이벤트 - 인증실패, 런타임예외 발생 - 를 포착하여 적절히 처리함으로써, 전반적인 보안 품질을 향상 시키는 것이다.

JEUS 보안 시스템은 보안 이벤트를 기반으로 한, 단순하지만 유연한 보안 감사 메커니즘을 특징으로 한다. 주요한 이벤트가 발생할 때 마다 - 인증 실패, 권한 부여 실패, 관심도가 높은 기타 이벤트 - 이벤트는 미리 등록된 이벤트 핸들러에 포착된다. 커스텀 핸들러 구현 클래스는 이에 적절하게 대응한다. 가령, 한 번에 연속적으로 너무 많이 인증에 실패할 경우, 해당 Subject 의 Credential 에 락을 걸도록 할 수 있다.

2.5.5 서비스와 SPI 개념

이전에도 언급했듯이, 보안 시스템에서 실제 보안 기능을 구현하고 있는 클래스를 서비스라고 한다. 다시 말해서, 서비스란 특정 보안 기능 - 인증, 권한 부여, 네트워킹, 기타 - 을 제공하는 SPI(Service Provider Interface)를 구현한 클래스이다.

"SPI"는 Service Provider Interface 의 약어로, jeus.security.spi 패키지에 포함되어 있는 추상클래스이다. 커스텀 보안 기능을 구현하기 위해, SPI 클래스를 다양하게 확장하고 구현할 수 있다. SPI 를 확장한 서브 클래스가 바로 서비스이다.

보안 시스템에서 서비스 인스턴스는 각각이 특정 보안 기능을 제공하는 독립적인 실체로 간주된다. 그러나, 종종 서비스는 다른 SPI 를 호출하기도 하기 때문에 서비스들간의 어느 정도 의존성은 존재한다.

서비스를 초기화하기 위해, 모든 서비스는 "key-value" 로 이루어진 한 쌍의 속성값을 전달받는다. 이러한 데이터는 설정 파일을 사용해서 저장해 둘 수도 있다.

모든 서비스는 선택사항으로, JEUS 관리 시스템에 보고할 JMX 빈을 정의하기도 한다. JMX MBean 은 보통 Service.getMBeanInfo() 메소드로부터 리턴받은 MBeanInfo 를 토대로 생성된다.

또한 모든 서비스는 생성 상태와 소멸 상태, 두 가지 상태를 가지고 있다 [그림 9]. 이 두 상태 간의 전이는 create()와 destroy()가 불려질 때 발생한다. 이 메소드를 호출하면, 실제 추상 메소드인 doCreate()와 doDestroy() 메소드가 호출된다. 이 추상 메소드들은 서비스 서브 클래스에서 구현되는데, 서비스를 초기화하고, 서비스를 정리하는 작업을 포함한다.

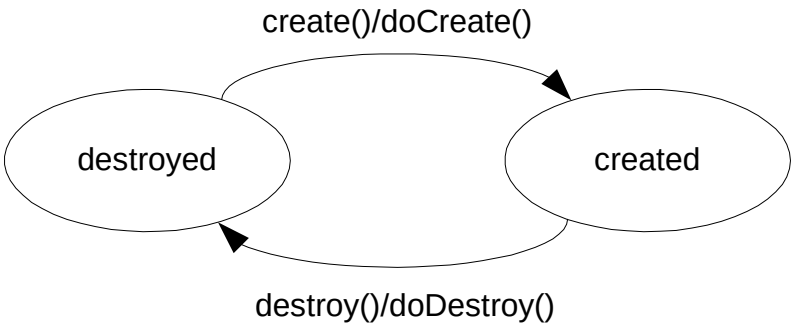

그림 9 서비스의 두 가지 상태

아래는 jeus.security.spi 패키지에 포함되어 있는 다양한 SPI 클래스와 Service 클래스들을 클래스 다이어그램을 사용해 보여주고 있다[그림 10].

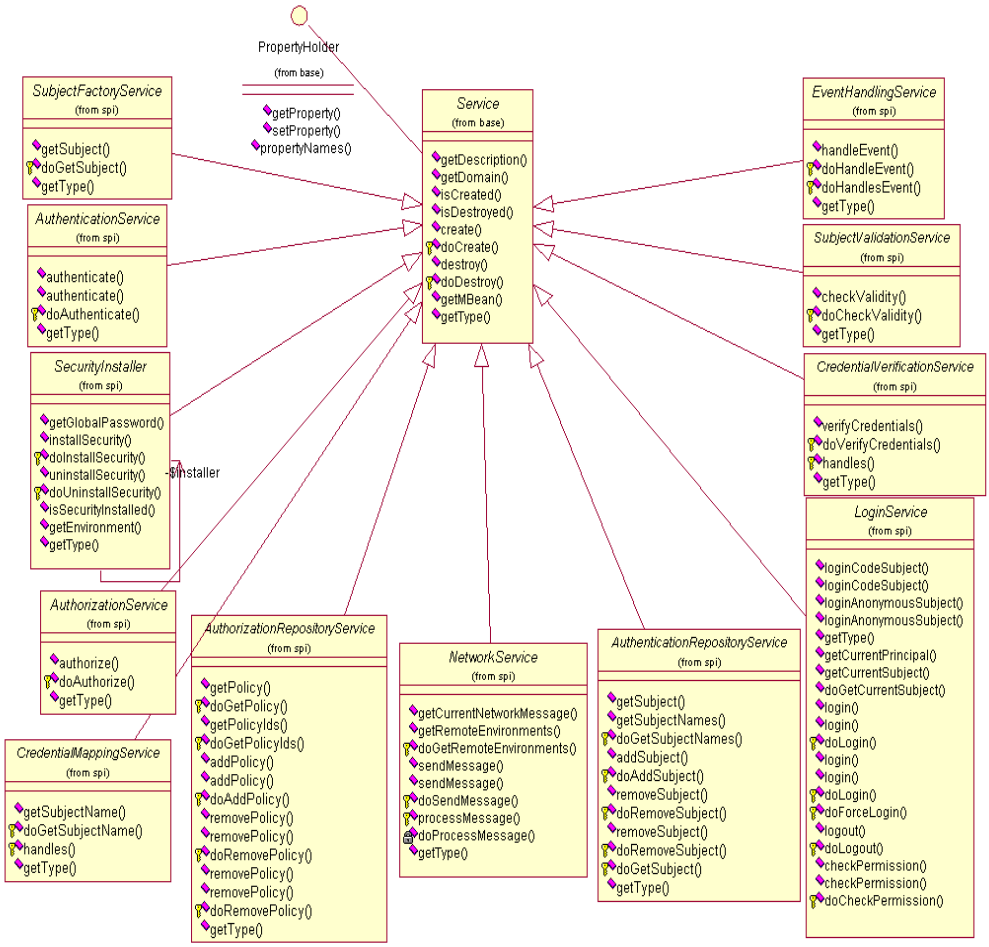

그림 10 Service 클래스와 SPI 서브 클래스들

런타임에, 서비스 인스턴스는 SecurityInstaller 싱글톤 클래스에 의해 생성된다. 단순한게 SecurityInstaller 를 구현한다면 서비스를 프로그램에서 직접 생성하는 코드만 작성하면 되겠지만, 더 유연하고 정교하게 서비스를 생성하는 방법이 있다. 서비스 클래스명과 속성 값을 특정 설정 파일에 저장해 놓고, 파일로부터 설정 정보를 읽어들여, 서비스 인스턴스를 만들고 초기화 하는 방법이다. 디폴트 SecurityInstaller 구현 클래스는 후자의 접근법을 도입하였으므로, 쉽고 유연한 방법으로 서비스를 보안 시스템에 추가할 수 있다.

주의: 단순히 JEUS 보안 시스템을 사용하기 위해라면, 서비스나 SPI 아키텍쳐를 모두 이해할 필요는 없다.

2.5.6 도메인 개념

보안 도메인(Security Domain) 이란 보안 서비스 인스턴스의 집합이라 할 수 있다. 하나의 보안 시스템에 여러 개의 도메인이 동시에 있을 수 있다.

도메인 개념의 배경에는 서로 다른 어플리케이션이나 JEUS 서브 시스템이 각각 별도의 보안 서비스를 사용할 수 있도록 하자는 데 있다. 도메인은 보안 서비스를 각 어플리케이션별로 분리하는 역할을 한다.

예로 들어서, JEUS 시스템만 사용하는 특별한 도메인을 설정할 수 있다. JEUS 가 사용하는 도메인을 SYSTEM_DOMAIN 이라고 하자. 이 도메인은 JEUS 를 관리하는데 사용하는 Subject 와 Policy 를 포함하고 있다. SYSTEM_DOMAIN 에 서버를 부트, 다운 할 수 있는 "administrator"라는 메인 Subject 를 설정할 수 있을 것이다.

또 다른 성격의 도메인으로, deploy 된 특정 J2EE 어플리케이션에서만 사용하는 도메인을 설정할 수 있다(이를 APP_DOMAIN 이라 하자). APP_DOMAIN 도 "administrator"라는 Subject 를 설정할 수 있지만, JEUS 의 "administrator"와는 Permission 이 다르다. 또한 도메인별로 Repository 메커니즘을 다르게 설정할 수 있다. 가령 SYSTEM_DOMAIN 이 Subject 를 XML 파일에 저장하는데 반해, APP_DOMAIN 은 원격지에 있는 데이터 베이스를 사용해서 Subject 를 읽고 쓰도록 설정할 할 수 있다.

아래 [그림 11]은 도메인 개념을 보여주고 있다.

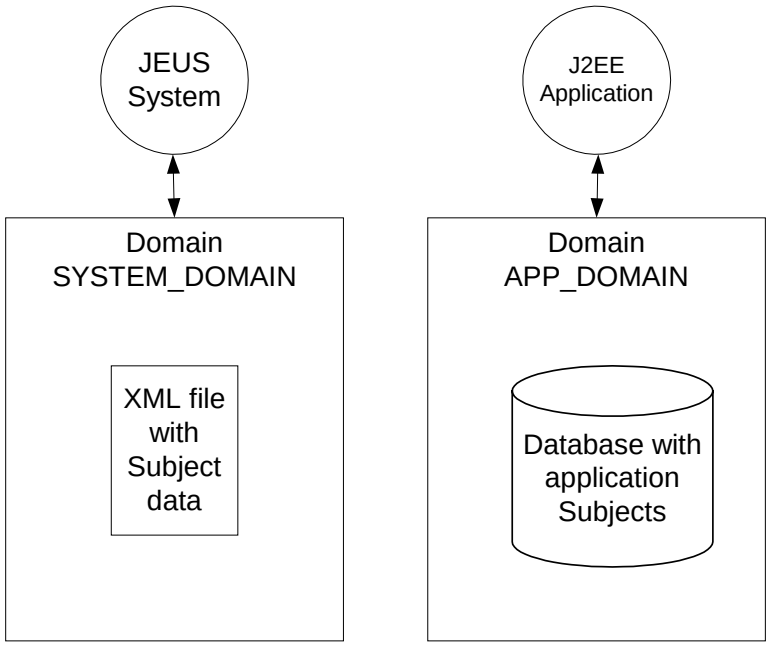

그림 11 다른 Application 과 다른 Subject Repository 를 사용한 두개의 도메인

도메인을 따로 설정할지는 어디까지나 선택 사항이다. 만약 도메인을 별도로 설정하지 않다면, 특별한 도메인인 SYSTEM_DOMAIN 이 사용된다.

주의: 도메인명은 모두 대문자로 쓰고 "_DOMAIN"으로 끝나는 것이 보통이다. 이것은 꼭 지켜야 되는 것은 아니고, 추천되는 명명법이다. 앞으로 이 문서에서 도메인이라 함은 모두 보안 도메인을 말하는 것이다. 보안 도메인은 J2EE 서버 도메인과는 아무런 상관도 없는 별개의 개념이다.

항상 두가지 디폴트 도메인이 JVM 에 존재한다.

- SYSTEM_DOMAIN: 표준 관리툴을 사용하여 JEUS Server 를 관리하는데 사용되는 도메인으로, 부트와 다운 같은 JEUS Server 를 관리하는 administrator 계정과 permission 을 포함하고 있다. 또한 J2EE 어플리케이션과 모듈에서 사용되며, JEUSMain.xml 에 도메인을 별도로 명시하지 않는 경우 사용되는 도메인이다.

- SHARED_DOMAIN: 특별한 도메인으로, 여기에 설정된 보안 서비스는 다른 모든 도메인과 공유된다. 따라서, SHARED_DOMAIN 은 모든 다른 도메인에서도 공통적으로 적용 가능한 보안 서비스를 포함하고 있어야 한다.

2.6 결론

JEUS 보안 시스템을 개괄적인 수준에서 살펴보았다. 구체적으로 JEUS 보안 시스템의 설계상 목표와 특징, 아키텍쳐, 보안관련 주요 개념 - "로그인", "사용자 인증", "권한 부여" "서비스", "도메인" - 을 다루었다.

다음 장에서는 JEUS 보안 시스템을 실제 어떻게 설정하는지 알아보겠다.

3 보안 시스템의 설정

3.1 소개

이 장에서는 보안 시스템을 실제로 어떻게 설치하고 설정하는지 알아 보겠다.

주의: 본 장에서 논하는 내용은 JEUS 가 디폴트로 제공하는 보안 시스템만을 다룬다. 만약 SecurityInstaller 등과 같은 보안 서비스를 디폴트가 아닌 다른 클래스로 설치하였다면, 본 장에서 설명한 내용이 정확하지 않을 수도 있다. 그런 경우에는 설치한 보안 서비스에 대한 문서를 참고해야 한다.

3.2 설정 개요

3.2.1 소개

본 절에서는 보안 시스템 설정과 관련된 기본적인 사항들을 간단하게 요약하겠다.

3.2.2 디폴트 보안 시스템 설정 과정

디폴트 보안 시스템을 설정하기 위해, 다음과 같은 순서를 따른다.

1. 보안 도메인들을 설정한다.
2. 각 보안 도메인별로 보안 서비스를 설정한다.
3. 각 도메인에 대한 Subjects (인증 데이터)를 설정한다.
4. 각 도메인에 대한 Policies(권한 부여 데이터)를 설정한다.
5. Subject 와 Policy 이외의 추가 사항을 설정한다.
6. J2SE SecurityManager 를 설정한다 (선택적).
7. JACC Provider 를 설정한다(선택적).

3.2.3 디폴트 보안 시스템의 디렉토리 구조

아래 [그림 12]는 디폴트 보안 시스템의 디렉토리 구조를 보여 주고 있다. 각 디렉토리에는 보안 시스템에서 사용하는 설정 파일들이 나열되어 있다.

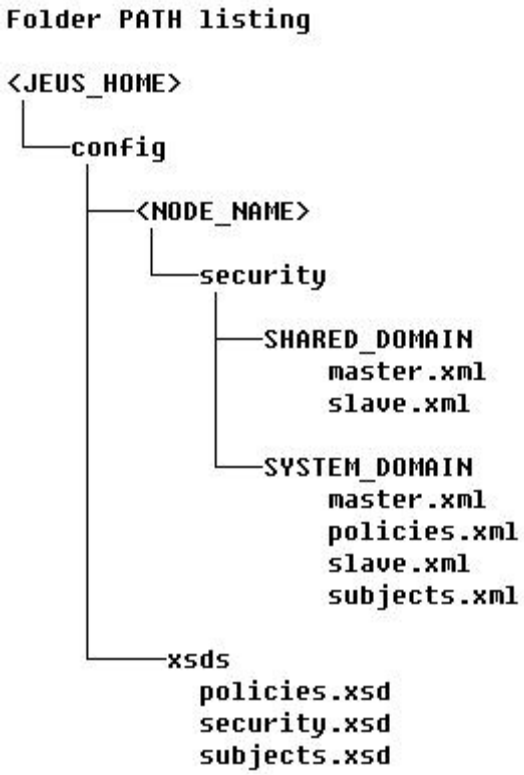

그림 12 디폴트 보안 시스템의 설정 파일 디렉토리 구조

주의: 설정 파일중에서 master.xml 과 slave.xml 이 존재지 않으면 기본으로 정해진 서비스들이 동작하게 된다. 즉, master.xml 및 slave.xml 이 반드시 필요하지는 않은 반면, SYSTEM_DOMAIN 에서 subjects.xml 및 policies.xml 은 반드시 필요하다.

3.2.4 결론

지금까지 JEUS 에서 보안 설정과 관련된 기본적인 사항을 간단히 살펴 보았다. 다음 절에서 위의 3.2.2 에서 간단하게 언급한 각 설정 단계들을 하나씩 자세히 살펴 보도록 하겠다.

3.3 보안 시스템 도메인 설정하기

디폴트 SecurityInstaller 를 사용하여, 새로운 보안 도메인을 설정하려면, JEUS_HOME\config\<node name>\security 패스 아래에 새로운 디렉토리를 생성해야 한다. 디렉토리명은 생성하려는 도메인의 이름과 일치해야 한다. 관례로 도메인명은 모두 대문자로 쓰고 "_DOMAIN"으로 끝맺는다. 예를들면 MY_DOMAIN 으로 명명할 수 있다. MY_DOMAIN 도메인을 생성하기 위해, 다음 명령어를 실행하도록 하자(MS Windows 명령프롬프트에서 실행한다).

```
c:\>mkdir c:\jeus\config\webhost\security\MY_DOMAIN
```

"webhost"는 실제 머신의 호스트 명으로 대체되어야 한다.

새로운 도메인 디렉토리를 생성한 후에, 그 안에 몇 가지 설정 파일들을 생성해 둔다. 가장 좋은 방법은 기존 도메인 디렉토리에서 설정 파일들을 그대로 복사해 와서, 필요에 따라 설정 파일을 변경하는 것이다. 예:

```
c:>copy c:\jeus\config\webhost\security\SYSTEM_DOMAIN\*.*
c:\jeus\config\webhost\security\MY_DOMAIN
```

(위 명령어는 모두 한 라인에 쓴다) 다음 절에서 복사된 각 설정 파일을 변경하는 방법을 설명한다.

디폴트로 JEUS_HOME\config\<node name>\security 패스 아래에 다음 두가지 도메인이 이미 존재하고 있다.

- SYSTEM_DOMAIN: JEUS Server 가 사용자 인증과 권한 체크를 위해 사용하는 도메인. 이 도메인은 JEUS 를 부트하고, 다운하는 등의 Permission 들과 JEUS 시스템 administrator 계정을 포함하고 있다. 어플리케이션이나 관리 툴에서 특별히 도메인을 설정해 놓지 않았을 때 적용되는 메인 보안 도메인.
- SHARED_DOMAIN: 이 도메인에 설정된 보안 서비스는 모든 다른 도메인에도 공통적으로 적용된다. 따라서, 모든 다른 도메인에도 공통적으로 적용해야 하는 보안 서비스가 필요하다고 판단될 때만, 수정되어야 한다. 이 도메인은 모든 다른 도메인에서 공유되어야 하므로, 디폴트로 네트워크 서비스를 포함하고 있다.

경고 : SHARED_DOMAIN 도메인을 수정할 때 세심한 주의가 요구된다. 특정 도메인에 유효한 보안 서비스가 다른 경우에는 문제를 발생시킬 수 있기 때문이다.

마지막으로, 서버가 시작할 때, 새로운 도메인을 인식할 수 있도록, JEUSMain.xml 을 수정해야 한다. 이에 대한 것은 4.5.3 절을 참조하기 바란다.

주의 1: 새롭게 생성된 도메인은 JEUS 를 다시 시작해야 적용된다.

주의 2: SYSTEM_DOMAIN, SHARED_DOMAIN 의 도메인은 JEUSMain.xml 의 설정과는 관계없이 항상 포함된다.

3.4 보안 시스템 서비스 설정하기

이전에 설명한 대로, JEUS 보안 시스템은 플러그(plug) 형태의 보안 서비스를 지원한다. 디폴트 SecurityInstaller 를 사용한다면, 각 도메인에서 로딩되는 보안 서비스는 해당 도메인 디렉토리 내의 다음 두 파일내에 설정되어 있다(도메인 디렉토리에 대해 이전 절을 참고하기 바란다).

- master.xml
- slave.xml

master.xml 은 JEUS Manager 의 JVM 에 설치되어 있는 보안 시스템에 적용된다. 따라서, Master 보안 시스템은 일차 보안 서버로 간주된다. 각 JEUS Manager 에는 오직 단 하나의 Master 보안 서버만 가능하다.

이차 보안 서버는 slave.xml 을 사용하여 설정된다. 이차 서버는 보통 JEUS 엔진 컨테이너의 JVM 에 설치된 보안 시스템이다. 여러 개의 Slave 서버는 단 하나의 Master 서버에 커넥션을 맺는다.

주의 위의 두 파일이 존재하지 않으면 기본으로 설정된 서비스들이 서비스하게 된다.

XML 파일 설정 방법은 JEUS_HOME\config\xsds 디렉토리내의 security.xsd 의 XML 스키마에 정의되어 있다. 내용은 다음과 같다.

최상위 태그는 **security-services** 이며, 이 태그는 다음 하위 태그를 가진다.

- **security-service** (0 개 이상) : 각 태그는 도메인에 추가되는 보안 서비스를 정의한다.
  - **Classname** (필수적) **:** 보안 서비스를 구현한 Java 클래스명이다. 이 클래스는 파라미터가 없는 디폴트 public 생성자를 가지고 있

어야 하며, jeus.security.spi 패키지의 SPI 클래스를 상속받거나, jeus.security.base.Service 클래스를 직접 상속해야 한다.

- **property** (0 개 이상) **:** jeus.security.base.PropertyHolder 인터페이스(jeus.security.base.Service 클래스가 구현한다)를 통해 보안 서비스에 name-value 쌍으로 속성을 설정할 수 있다. 속성은 각 보안 서비스를 초기화하는데 사용된다. 다음 두 개의 하위 태그를 가진다.
  - **Name**: 속성명
  - **Value** (선택) : 속성명에 해당하는 String 속성 값.

다음은 보안 설정 파일의 예제이다.

<<*master.xml*>>

```
<?xml version="1.0"?>

<security-services xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <security-service>
        <classname>
            jeus.security.impl.login.ServerLoginService
        </classname>
        <property>
            <name>prop1</name>
            <value>prop1value</name>
        </property>
        <property>
            <name>prop2</name>
        </property>
    </security-service>
    <security-service>
        <classname>
            jeus.security.impl.atn.DefaultAuthenticationService
        </classname>
    </security-service>
    <security-service>
        <classname>
      jeus.security.impl.verification.PasswordVerificationService
        </classname>
```

```
    </security-service>
    . . .
</security- services>
```

JEUS 가 디폴트로 제공하는 보안 서비스에는 어떤 것이 있는지는 자세히 알고 싶다면, 부록 F 를 참고하기 바란다. 만약 커스텀 보안 서비스를 작성하고 싶다면, 9 장을 참고하기 바란다.

3.4.1 디폴트 SecurityInstaller 를 교체하기

새로운 SecurityInstaller 를 구현해서 디폴트 installer 를 교체하는 방법이 있다. 자세한 것은 9 장을 참조하기 바란다.

만약 9 장에서 설명한 대로, 자신만의 SecurityInstaller 를 구현하였다면, 해당 보안 서비스를 master.xml 과 slave.xml 에 설정하는 것이 아니라, JEUS 시작 스크립트 (jeus.cmd)와 JEUSMain.xml 에 다음과 같이 설정해 주어야 한다.

```
-Djeus.security.installer.classname=<fully qualified class name>
```

예로 자신이 구현한 클래스명을 “mypackage.MySecurityInstaller”라고 하자.

<<jeus.properties.cmd>>

```
. . .
rem setup JAVA_ARGS.
SET JAVA_ARGS=-Djeus.security.installer.classname=mypackage.MySecur
ityInstaller
. . .
```

<<jeus.properties>>

```
. . .
# setup JAVA_ARGS.
JAVA_ARGS=-Djeus.security.installer.classname=mypackage.MySecurityI
nstaller
. . .
```

<<JEUSMain.xml>>

```
<?xml version="1.0"?>
<jeus-system xmlns=“http://www.tmaxsoft.com/xml/ns/jeus”>
    <node>
```

```
        <name>johan</name>
            <engine-container>
                <name>container1</name>
                <!-- engine container JVM option -->
                <command-option>
                   …
                   -Djeus.security.installer.classname=
                   mypackage.MySecurityInstaller
                </command-option>
                <engine-command>
                    <type>servlet</type>
                    <name>engine1</name>
                    <startup-mode>xml</startup-mode>
                </engine-command>
    . . .
```

동일한 설정은 JEUS_HOME\bin 디렉토리내의 모든 JEUS 스크립트에도 적용되어야 한다.

주의: 보통은 디폴트 SecurityInstaller 를 교체할 필요가 없다.

3.5 보안 시스템 Subject 설정하기

디폴트 보안 설정에서, Subject 데이터는 "subjects.xml"로 불리는 XML 파일에서 읽어 들인다. 이 파일은 JEUS_HOME\config\<node name>\security\<domain name>\subjects.xml 경로에 있다. 여기서 <node name>은 호스트 머신의 이름이고, <domain name>은 Subject 가 속한 보안 도메인의 이름을 나타낸다.

subjects.xml 의 XML 스키마는 subjects.xsd 이며, JEUS_HOME\config\xsds 경로에 있다.

이 파일에는 0 개 이상의 **subject** 태그가 포함되어 있으며, 각각은 Subject(user)를 나타내고 있다. 각 **subject** 태그는 다음과 같은 하위 태그를 가지고 있다.

- **description** (선택) : user 에 대한 설명 (문자열)
- **principal** (0 개 이상) : Subjet 에 대한 다양한 신원을 나타낸다(예: 사용자 명). **principal** 은 다음 두가지 태그로 정의된다.

- **classname** (선택) : java.security.Principal 을 구현한 클래스명. 설정되어 있지 않으면, 디폴트로 jeus.security.resource.PrincipalImpl 가 적용된다.
- **name** (필수) **:** Principal 명

참고: 최소한 하나의 **principal** 태그가 정의되어 있어야 한다. 처음에 정의된 Principal 이 Subject 에 대한 메인 Principal 이다. 메인 Principal 은 Subject 에 대한 id 처럼 행동하며, 시스템에 로그인할 때 사용자가 직접 입력해야 한다(따라서 처음으로 정의된 Principal 을 해당 Subject 에 대한 "user name" 또는 "user id"라고 생각하면 된다). 이외의 추가적인 Principal 은 선택 사항 이며, 보통 Subject 에 고유한 것이 아니라, 그룹 Principal 을 나타낸다.

- **credential** (0 개 이상, 선택적) **:** CredentialFactory 를 나타낸다. 각 credential 태그는 다음 태그로 정의된다.
  - **classname** (선택적)**:** 런타임에 Credential 을 생성하는 jeus.security.base.CredentialFactory 를 구현한 클래스명. 별도로 설정되어 있지 않으면, 디폴트로 jeus.security.resource.Password 인스턴스를 생성하는 jeus.security.resource.PasswordFactory 가 적용된다.
  - **property** (0 개이상) **:** 속성 (key-value 쌍)을 나타내며, jeus.security.base.CredentialFactory 를 초기화 하는데 사용된다. 가령, jeus.security.resource.PasswordFactory 는 "password"라는 이름의 속성이 필요하고, 속성 값은 Base 64 로 인코딩된 것이어야 한다 (아래 예제를 참고하기 바란다).

subjects.xml 의 예제 :

<<*subjects.xml*>>

```
<?xml version="1.0"?>

<subjects xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
     <subject>
        <description>The j2ee Subject</description>
        <principal>
            <classname>mypackage.MyPrincipalImpl</classname>
            <name>j2ee</name>
        </principal>
```

```
        <principal>groupA</principal>
        <principal>groupB</principal>
        <credential>
            <classname>
                jeus.security.resource.PasswordFactory
            </classname>
            <property>
                <name>password</name>
                <value>ajJlZQ==</value>
            </property>
        </credential>
    </subject>
    <subject>
        <principal>
            <!-- jeus.security.resource.PrincipalImpl implied
                 as Principal classname -->
            <name>javajoe</name>
        </principal>
        <credential>
            <!-- jeus.security.resource.PasswordFactory implied
                 as CredentialFactory classname -->
            <property>
                <name>password</name>
                <value>amF2YWpvZQ==</value>
            </property>
        </credential>
    </subject>
</subjects>
```

일반 문자열을 Base 64 로 인코딩하기 위해서, JEUS_HOME\bin\base64 스크립트를 사용한다. 자세한 사항은 부록 B 를 참고하기 바란다.

3.5.1 커스텀 Credential 구현 및 설정하기

앞서 보았듯이, Credential 을 subjects.xml 에 추가하려 할 때마다, jeus.security.base.CredentialFactory 인터페이스를 구현한 클래스를 지정해야 했다(선택사항으로, 지정하지 않으면 jeus.security.resource.PasswordFactory 가 디폴트로 사용된다). CredentialFactory 는 런타임에 Subject 에 대한 실제 Credential 을 생성하는데 사용된다. 따라서, 커스텀 Crdential 을 사용하고자 한

다면, 해당 Credential 을 생성하는 로직을 가진 CredentialFactory 클래스를 구현해야 한다.

커스텀 CredentialFactory 는 CredentialFactory 인터페이스를 구현한 것으로, 반드시 다음 두가지 메소드를 구현해야 한다.

- getCredential(): 이 메소드는 java.lang.Object 타입의 실제 Credential 을 리턴한다. Credential 은 어떤 종류의 Object 라도 상관없으며, 간단한 패스워드이거나, 인증서 저장소에 저장된 디지털 인증서일 수도 있다.

- isPublicCredential(): getCredential() 메소드로부터 리턴된 Credential 의 기밀성을 유지할 필요가 없다면(예, 디지털 인증서) true 를, 기밀성을 유지할 필요가 있다면 (예, 패스워드) false 를 리턴한다.

추가적으로, CredentialFactory 인터페이스는 jeus.security.base.PropertyHolder 인터페이스를 상속하므로, CredentialFactory 를 구현하는 클래스는 PropertyHolder 인터페이스를 구현해야만 한다.

PropertyHolder 인터페이스는 CredentialFactory 에 필요한 추가적인 정보를 제공하는데 사용된다. 가령, CredentialFactory 가 디지털 인증서 Credential 을 생성하려면 인증서 저장소의 위치를 알고 있어야 한다. 마찬가지로 비밀번호 Credential 을 생성하려면, 비밀번호를 알고 있어야 한다. 이런 정보를 subjects.xml 의 <credential> 태그에 등록하면, PropertyHolder 인터페이스의 메소드를 통해서 CredentialFactory 로 정보가 전달된다.

PropertyHolder 인터페이스에서 구현해야 하는 메소드는 다음과 같다.

- getProperty(String name): 해당 속성 값을 리턴한다. 이 메소드는 속성 값이 설정되어 있으면 java.lang.String 타입을, 설정되어 있지 않으면 null 을 리턴한다.

- propertyNames(): 현재 설정된 속성명을 Enumeration 타입으로 리턴한다.

- setProperty(String name, String value): 주어진 이름과 값 쌍으로 속성을 설정한다.

다음 예는 패스워드 Credential 을 리턴하는 매우 간단한 코드를 보여주고 있다.

<<*MyPasswordCredentialFactory.java*>>

```
package mypackage;
```

```
import jeus.security.base.CredentialFactory;
import jeus.security.base.CredentialFactoryException;
import jeus.security.resource.PasswordFactory;

public class MyPasswordCredentialFactory extends PasswordFactory
implements CredentialFactory {
    public static final String PASSWORD_PROPERTY_KEY = "password";

    public MyPasswordCredentialFactory() {}

    public MyPasswordCredentialFactory(String password) {
        setProperty(PASSWORD_PROPERTY_KEY, password);
    }

    public String getPassword() {
        return getProperty(PASSWORD_PROPERTY_KEY);
    }

    public Object getCredential() throws
                            CredentialFactoryException {
        String password = getPassword();
        if (password == null) {
            throw new CredentialFactoryException("Error creating
Password: the password property is null");
        }
        return new MyPasswordCredential(password);
    }
    public boolean isPublicCredential() {
        return false;
    }
}
```

위의 예제에서, 실제 패스워드를 MyPasswordCredentialFactory 에 넘겨주기 위해 "password"라는 속성을 사용하고 있으며, MyPasswordCredentialFactory 는 실제 패스워드를 사용하여 MyPasswordCredential 인스턴스를 생성하고 있다. mypackage.MyPasswordCredential 의 소스 코드는 아래에 제시되어 있다.

<<MyPasswordCredential.java>>

```
package mypackage;
```

```
import java.io.*;
import jeus.security.resource.Password;

public class MyPasswordCredential extends Password {
    private String password = null;

    public MyPasswordCredential(String password) {
        super(password);
        this.password = password;
    }

    public boolean equals(Object another) {
        if (another == null || !(another instanceof Password)) {
            return false;
        }
        Password pw = (Password) another;
        if (pw.password == null || this.password == null) {
            return false;
        }
        return this.password.equals(pw.password);
    }

    public int hashCode() {
        if (this.password == null) {
            return 0;
        }
        return this.password.hashCode();
    }
}
```

새로 생성된 Credential 을 특정 Subject 에 설정하기 위해서, 다음과 같이 subjects.xml 을 변경한다.

<<subjects.xml>>

```
<?xml version="1.0"?>

<subjects xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
     <subject>
        <principal>
```

```
            <name>j2ee</name>
        </principal>
        <credential>
            <classname>
                mypackage.MyPasswordCredentialFactory
            </classname>
            <property>
                <name>password</name>
                <value>mypass</value>
            </property>
        </credential>
    </subject>
    . . .
</subjects>
```

위에서는 subjects.xml 의 일부분을 보여 주고 있는데, 메인 Principal 이 "j2ee" 인 Subject 가 MyPasswordCredentialFactory 클래스를 통해 MyPasswordCredential 인스턴스와 결합된다는 것을 나타낸다. 실제 패스워드인 "mypass"는 "password" 속성을 사용하여, MyPasswordCredentialFactory 클래스로 전달된다.

주의 : 커스텀 Credential 을 구현하였으면, 해당 Credential 을 인증할 수 있는 인증 서비스도 함께 제공되어야 한다. 이 서비스는 jeus.security.spi.CredentialVerificationService 인터페이스를 구현하고 보안 시스템에 적절히 설정함으로써 제공될 수 있다. 자세한 내용은 9 장을 참고하기 바란다.

3.5.2 DB 를 이용한 Subject 설정하기

데이터베이스를 이용하여 Subject 를 설정하려면 다음과 같이 master.xml 이나 slave.xml 에 지정하여야 한다.

<<*master.xml*>>

```
<?xml version="1.0"?>

<security-services xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <security-service>
        <classname>
jeus.security.impl.atnrep.DBPersistedDistributedMemoryAuthenticatio
 nRepositoryService
        </classname>
```

```
        <property>
                <name>driver</name>
                <value>oracle.jdbc.OracleDriver</value>
        </property>
        <property>
                <name>url</name>
                <value>
                    jdbc:oracle:thin:@127.0.0.1:1521:ORA9I
                </value>
        </property>
        <property>
                <name>username</name>
                <value>scott</value>
        </property>
        <property>
                <name>password</name>
                <value>dGlnZXI=</value>
        </property>
    </security-service>
     . . .
</security- services>
```

우선, 기본으로 설정되어있는 AuthenticationRepositoryService 인 jeus.security.impl.atnrep.XMLPersistedDistributedMemoryAuthenticationRepositoryService 를 계속 사용을 하면 두 개의 저장공간을 가지게 된다. subjects.xml 을 사용하지 않으려면 이 기본설정의 서비스를 삭제해야 한다.

DB 를 사용하는 AuthenticationRepositoryService 를 사용하려면 DB 에 접근하는 JDBC driver 와 접근을 하기위한 정보인 URL, username 그리고 password 가 필요하다. 본 예제에서는 Oracle 에 접근하기 위한 정보로 password 는 base64 로 encode 된 문자가 입력된다.

DB 에서 쓰이는 테이블은 다음과 같이 구성된다.

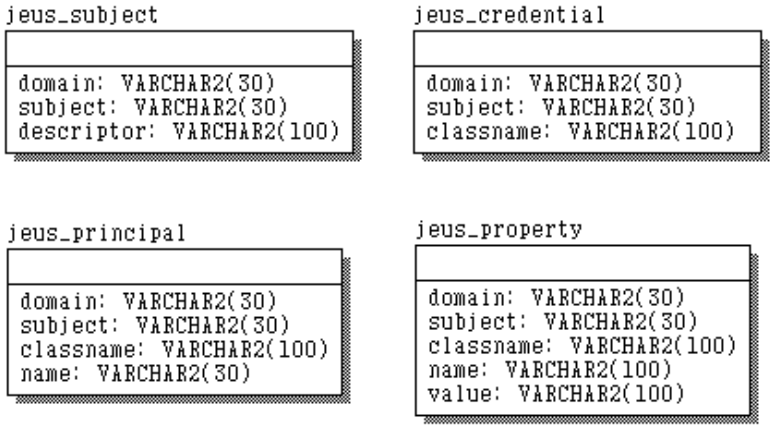

그림 13 Subject 를 저장하기 위한 DB Table 구조

DB 에 테이블이 존재하지 않는다면 자동으로 테이블을 만들고 사용자 명이 jeus, 패스워드가 jeus 인 사용자가 최초로 만들어지게 된다. 만약 사용자를 추가하고 싶다면 웹 매니저를 이용하여 사용자를 추가하고 "save" 버튼을 눌러서 사용자를 추가한다.

3.6 보안 시스템 Policy 설정하기

디폴트 보안 설정시, Policy 데이터 (권한 부여 데이터) 는 policies.xml 파일에서 읽어온다. 이 파일은 JEUS_HOME\config\<node name>\security\<domain name>\policies.xml 경로에 있다. 여기서 <node name>은 실제 호스트 머신의 이름이어야 하고, <domain name>은 Policy 가 속해 있는 도메인명이어야 한다.

policies.xml 의 XML 스키마는 policies.xsd 로써, JEUS_HOME\config\xsds 디렉토리 내에 있다.

policies.xml 은 0 개 이상의 **policy** 태그로 구성되어 있으며, 각 태그는 개별 Policy (권한 부여 데이터)를 나타낸다. 각 policy 태그는 다음 태그들로 구성되어 있다.

- **role-permissions** (0 또는 1) **:** principal-to-role 매핑들에 대한 정보를 제공한다. 하위 태그로 0 개 이상의 **role-permission** 태그를 가지고 있다. **role-permission** 태그는 다음의 태그로 구성되어 있다.
  - **principal** (0 개 이상)**:** 현재 role-permission 을 소유하고 있는 Principal 명

- **role** (1 개, 필수적) **:** Role 명을 나타낸다.
- **actions** (선택적) **:** Role 에 대한 액션을 나타낸다.
- **classname** (선택적) **:** Role Permission 으로 사용될 java.security.Permission 를 구현한 Java 클래스명. 생략되면, jeus.security.resource.RolePermission 이 디폴트로 사용된다. 커스텀 Permission 을 설정하고 구현하는 방법은 3.6.1 절을 참고하기 바란다.
- **excluded** (선택적) **:** 값이 없는 empty 태그(<xxx/>) 이다. 설정되어 있으면 해당 Role Permission 이 배제된다(즉 Permission 이 암시하는 Role 에 아무도 접근할 수가 없다).
- **unchecked** (선택적) **:** empty 태그이다. 설정되어 있으면 Role Permission 이 체크되지 않는다(즉 Permission 이 암시하는 Role 에 누구나 접근할 수 있다).

- **resource-permissions** (0 개 이상) : role-to-resource 에 대한 정보를 가지고 있다.
  다음의 태그로 구성되어 있다.
  - **context-id** (선택적) **:** context id 를 나타낸다(문자열). context 란 권한 부여 체크가 일어나는 범위를 나타낸다. JEUS 시스템이 JEUS 리소스에 대해 사용하는 디폴트 context id 는 "default"이다.
  - **resource-permission** (0 개 이상)**:** 다음의 항목으로 구성되어 있다 .
    - **role** (0 개 이상) **:** 현재 resource-permission 을 소유하고 있는 Role 명
    - **resource** (1 개, 필수적) **:** 리소스 명을 나타낸다.
    - **actions** (선택적) **:** 리소스에 대한 액션을 나타낸다.
    - **classname** (선택적) **:** Resource Permission 으로 나타내는 java.security.Permission 을 구현한 클래스명. 생략하면 jeus.security.resource.ResourcePermission 이 디폴트로 사용된다. 커스텀 Permission 을 생성하고 설정하는 방법은 3.6.1 절을 참고하기 바란다.

- **excluded** (선택적) : 값이 없는 empty 태그이다. 설정하면, 해당 Resource Permission 이 배제된다(즉 Permission 이 암시하는 리소스는 누구도 접근할 수 없다).

- **unchecked** (선택적) **:** 값이 없는 empty 태그이다. 설정하면, 해당 Resource Permission 을 체크하지 않는다 (즉 Permission 이 암시하는 리소스는 누구나 접근할 수 있다).

policies.xml 의 예 :

<<policies.xml>>

```
<?xml version="1.0"?>

<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <role-permissions>
            <role-permission>
                <principal>johan</principal>
                <role>administrator</role>
            </role-permission>
            <role-permission>
                <principal>johan</principal>
                <role>teller</role>
                <actions>9:00-17:00</actions>
                <classname>
           jeus.security.resource.TimeConstrainedRolePermission
                </classname>
            </role-permission>
        </role-permissions>
        <resource-permissions>
            <context-id>default</context-id>
            <resource-permission>
                <role>teller</role>
                <resource>bankdb</resource>
                <actions>select, update</actions>
            </resource-permission>
            <resource-permission>
                <role>administrator</role>
                <resource>jndi</resource>
                <actions>modify, delete</actions>
```

```
            </resource-permission>
            <resource-permission>
                <resource>jeus.*</resource>
                <actions>boot, down</actions>
                <excluded/>
            </resource-permission>
            <resource-permission>
                <resource>jndi</resource>
                <actions>lookup</actions>
                <unchecked/>
            </resource-permission>
            <resource-permission>
                <role>administrator</role>
                <resource>jeus.security.*</resource>
                <actions>*</actions>
            </resource-permission>
        </resource-permissions>
    </policy>
</policies>
```

주의 : 보통 policies.xml 은 JEUS Server 리소스에 대한 Permission (JNDI, JMS, Security Server 등)을 설정하는데 사용되고, J2EE 어플리케이션과 J2EE 모듈에 대한 Permission 을 설정하는데는 사용하지 않는다. 대신 J2EE 어플리케이션과 모듈에 Permission 을 설정하기 위해, 다양한 DD 파일을 사용한다. 자세한 사항은 4 장을 참고하기 바란다.

3.6.1 커스텀 Permission 구현 및 설정하기

이전 절에서 보았듯이, Permission(Role Permission 또는 Resource Permission)을 policies.xml 파일에 추가할 때마다, Permission 을 나타내는 Java 클래스명을 설정했다. 해당 클래스는 java.security.Permission 추상 클래스를 확장한 것이다. 자신만의 java.security.Permission 구현 클래스를 만들어, policies.xml 에 <classname>태그로 설정해 두면, 커스텀 Permission 을 직접 만들 수 있다.

지금부터 어떻게 커스텀 Permission 을 만들 수 있는지 자세히 살펴보도록 하자. 다음 요구사항을 가진 새로운 커스텀 Role Permission 을 개발해 보겠다.

새로운 Role Permission 은 원래의 Role 외에 다음과 같은 두 가지 조건이 충족되는 상황에서 또 다른 Role 을 암시할 수 있다.

1. 또 다른 Role 도 원래의 Role 과 동일한 이름을 가지고 있다.

2. 현재 시간이 특정 시간 범위 (오전 9 시부터 오후 5 시까지) 내에 있을 때만 또 다른 Role 을 암시할 수 있다.

가령, 이러한 Role Permission 은 뱅킹 어플리케이션에서 "peter"라는 Principal 이 오전 9 시부터 오후 5 시까지 업무 시간 동안만 "teller"라는 Role 을 가질 수 있도록 만들 수 있다.

이를 위해 두가지 작업이 필요하다.

1. 커스텀 Permission 을 생성한다.
2. policies.xml 에 해당 Permission 을 사용하도록 설정한다.

아래는 위의 요구사항을 만족하는 커스텀 Permission 클래스를 구현한 것이다. 자세한 코드는 생략한다:

```
package jeus.security.resource;

import java.security.Permission;
import java.util.Calendar;
import java.util.StringTokenizer;

/**
 * A time-constrained Role permission.
 * <p>
 * With this permission implementation you can express
 * things such as "X can only be in the role Y between
 * 09:00 AM to 05:00 PM".
 */
public class TimeConstrainedRolePermission extends RolePermission {
    private String timeConstraint = "*";
    private int startTime = Integer.MIN_VALUE;
    private int endTime = Integer.MAX_VALUE;

    public TimeConstrainedRolePermission(String roleName) {
        this(roleName, "*");
    }

    public TimeConstrainedRolePermission (String roleName,
                                          String timeConstraint) {
```

```
        super(roleName);
        if (timeConstraint != null) {
            this.timeConstraint = timeConstraint;
            parseTimeConstraint();
        }
    }

    private void parseTimeConstraint() {
        . . .
    }

    public boolean equals(Object anotherObject) {
        . . .
    }

    public int hashCode() {
        . . .
    }

    public String getActions() {
        return timeConstraint;
    }

    public boolean implies(Permission anotherPermission) {
        if (this.timeConstraint.equals("*")) {
            return super.implies(anotherPermission);
        } else {
            Calendar cal = Calendar.getInstance();
            int curHour = cal.get(Calendar.HOUR_OF_DAY);
            int curMinute = cal.get(Calendar.MINUTE);
            int now = curHour * 60 + curMinute;
            if (now >= this.startTime && now <= this.endTime) {
                return super.implies(anotherPermission);
            } else {
                return false;
            }
        }
    }
}
```

위에서 보듯이, 해당 클래스는 jeus.security.resource.RolePermission 을 상속한다. jeus.security.resource.RolePermission 는 다시 java.security.Permission 을 상속한 것이다. 이는 코드 재사용성을 높이기 위한 구조이다. TimeConstrainedRolePermission 을 상속해 또 다른 java.security.Permission 을 만드는 것도 가능하다.

위의 소스 일부에서 보듯이, 두가지 타입의 생성자가 있다. 첫번째는 (role name)이란 하나의 파라미터만 받고 있고, 두 번째는 (role name, time constraint) 두 개의 파라미터를 받고 있다. java.security.Permission 에서, 첫번째 파라미터는 "name"을 뜻하고, 두 번째 파라미터는 "actions"를 뜻한다.

이 클래스의 경우, "actions" 파라미터는 Permission 에 대한 유효 시간을 나타내며, “09:00-17:00”라는 값을 가진다. "name"은 "administrator" 또는 "teller"와 같은 Role 명을 나타낸다. 몇몇 Permission 구현 클래스는 "actions" 파라미터를 받는 두 번째 타입의 생성자가 생략되기도 한다.

소스의 핵심은 "implies(Permission anotherPermission)" 메소드로써, 현재 시스템 시간이 주어진 유효 시간내에 있는지 체크한 다음, 만약 그렇다면 super.implies() 메소드를 호출하고, 그렇지 않으면 "false"를 리턴한다. 모든 implies(..)메소드는 boolean 값을 리턴하게 되어 있는데, 해당 Permission 이 파라미터로 넘어온 Permission 을 암시하는지를 나타낸다.

implies() 메소드와 java.security.Permission 추상 클래스에 대한 자세한 정보는 J2SE Javadoc 문서를 참고하기 바란다. jeus.security.resource.RolePermission, jeus.security.resource.TimeConstrainedRolePermission, jeus.security.resource.ResourcePermission 에 대한 자세한 정보는 부록 J 와 JEUS Security Javadoc 에서 jeus.security.resource 패키지를 참고하기 바란다.

지금까지 java.security.Permission 커스텀 클래스를 구현해 보았다. "javac"로 컴파일하고 JEUS Server 의 클래스 패스에 해당 클래스 경로를 잡아 주도록 하자.

마지막으로, 커스텀 Permission 클래스를 policies.xml 파일에 설정한다. policies.xml 은 다음과 같다.

<<policies.xml>>

```
<?xml version="1.0"?>

<policies xmlns=“http://www.tmaxsoft.com/xml/ns/jeus”>
    <policy>
        <role-permissions>
            <role-permission>
                <principal>peter</principal>
```

```
                <role>administrator</role>
                <actions>9:00-17:00</actions>
                <classname>
           jeus.security.resource.TimeConstrainedRolePermission
                </classname>
            </role-permission>
        </role-permissions>
        <resource-permissions>
            <context-id>default</context-id>
            <resource-permission>
                <role>administrator</role>
                <resource>jeus.*</resource>
                <actions>*</actions>
            </resource-permission>
       </resource-permissions>
    </policy>
</policies>
```

위의 XML 은 다음을 나타내고 있다.

"peter"라는 User 는 "administrator" Role 을 오전 9 시부터 오후 5 시까지 업무시간 동안만 부여 받는다. "administrator" Role 은 모든 JEUS 리소스('jeus.*') 에 대해 모든 액션(*)을 실행할 수 있는 권한을 가지고 있다. 따라서, peter 는 업무시간 동안만 모든 JEUS 리소스에 대한 모든 권한을 행사할 수 있다."

주의 : 위의 스키마는 J2EE 어플리케이션과 모듈에서 JEUS DD 파일을 설정할 때도 그대로 적용된다. JEUS DD 에서 커스텀 Permission 을 사용하는 방법은 4 장을 참고하기 바란다.

주의 2: 본질적으로 위의 스키마는 예제에서 보여준 principal-to-role Permission 에뿐 아니라, role-to-resource Permission 에도 동일하게 적용된다.

3.6.2 Database 를 이용한 Policy 설정하기

데이터 베이스를 이용하여 policy 를 설정하려면 다음과 같이 master.xml 이나 slave.xml 에 지정하여야 한다.

<<*master.xml*>>

```
<?xml version="1.0"?>

<security-services xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```

```
    <security-service>
       <classname>
jeus.security.impl.atnrep.DBPersistedDistributedMemoryAuthorization
  RepositoryService
       </classname>
       <property>
               <name>driver</name>
               <value>oracle.jdbc.OracleDriver</value>
       </property>
       <property>
               <name>url</name>
               <value>
                   jdbc:oracle:thin:@127.0.0.1:1521:ORA9I
               </value>
       </property>
       <property>
               <name>username</name>
               <value>scott</value>
       </property>
       <property>
               <name>password</name>
               <value>dGlnZXI=</value>
       </property>
   </security-service>
    . . .
</security- services>
```

우선, 기본으로 설정되어있는 AuthorizationRepositoryService 인 jeus.security.impl.atnrep.XMLPersistedDistributedMemoryAuthorizationRepositoryService 를 계속 사용을 하면 두 개의 저장공간을 가지게 된다. policies.xml 을 사용하지 않으려면 이 기본설정의 jeus.securety.impl.atnrep.XMLPersistedDistributed-MemoryAuthorizationRepositoryService 를 삭제해야 한다.

DB 를 사용하는 AuthenticationRepositoryService 를 사용하려면 DB 에 접근하는 jdbc driver 와 접근을 하기위한 정보인 url, username 그리고 password 가 필요하다. 본 예제에서는 Oracle 에 접근하기 위한 정보로 password 는 base64 로 encode 된 문자가 입력된다.

DB 에서 쓰이는 테이블은 다음과 같이 구성된다.

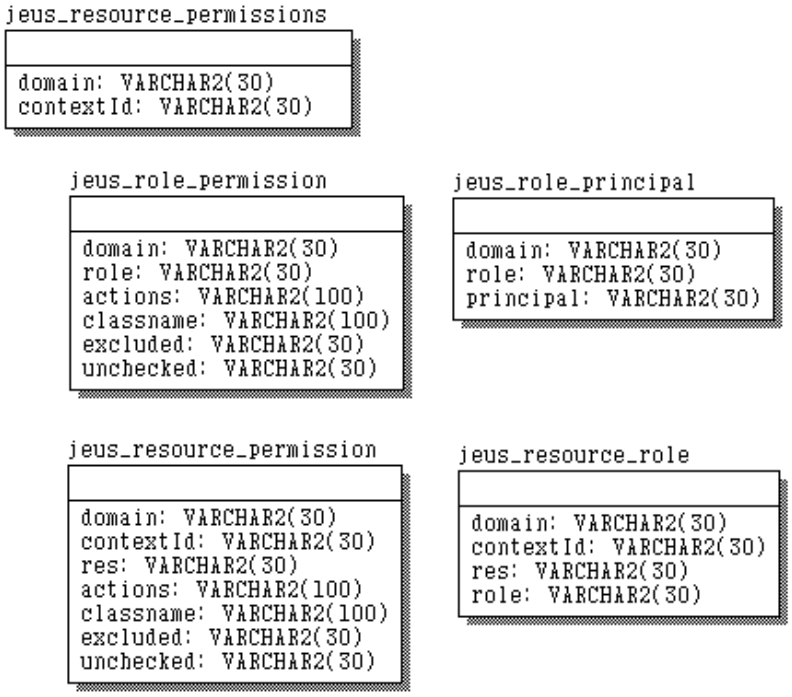

그림 14 Policy 를 저장하기 위한 DB Table 구조

DB 에 테이블이 존재하지 않는다면 자동으로 테이블을 만든다. 기본적인 Policy 는 “SYSTEM_DOMAIN”에 “administrator”라는 Role 을 가지게 되고, “jeus.*” 에 대한 Resource 권한을 가진다. “administrator” Role 포함된 principal 은 3.5.2 절에서 설명한 jeus 이다. 만약 policy 를 추가하고 싶다면 웹 매니저를 이용하여 policy 를 추가하고 “save” 버튼을 눌러서 policy 를 추가한다

3.7 추가 항목 설정하기

이 절에서는 Subject 와 Policy 이외의 추가 항목을 설정하는 방법을 알아 보겠다.

3.7.1 전역 시스템 패스워드 설정하기

클러스터로 묶여 있는 모든 보안 서버들은 서로간의 신뢰를 확보하기 위해 공통의 전역 시스템 패스워드를 사용한다.

참고: 보안 서버는 크게 두가지 종류가 있는데, JEUS Manager 의 VM 에 설치되어 있는 보안 서버와(Master 보안 서버 또는 1 차 보안 서버)와 엔진 컨테이너 VM 별로 설치되어 있는 보안 서버 (slave 보안 서버 또는 2 차 보안 서버)가 있다. 이들을 보안 클러스터로 묶었을 경우, 상호간의 신뢰를 확보하기 위해 VM 에 동일한 전역 시스템 패스워드를 설정해 주어야 한다.

시스템 패스워드는 클러스터로 묶여 있는 모든 JEUS VM 에서 동일해야 하며, 다양한 목적으로 사용된다. JVM 에서 시스템 패스워드를 설정하려면, 다음 Java 시스템 속성을 사용한다.

```
-Djeus.security.globalPassword=<password>
```

이 속성은 JEUS_HOME\bin 디렉토리내의 JEUS 구동 스크립트(가령 Window 에서 jeus.cmd)와 JEUSMain.xml 에 다음과 같이 설정한다. 예제 :

<<jeus.cmd 또는 jeus>>

```
. . .
java -server . . .
      –Djeus.security.globalPassword=syspass
      jeus.server.JeusBootstrapper %*
```

<<JEUSMain.xml>>

```
<?xml version="1.0"?>
<jeus-system xmlns=“http://www.tmaxsoft.com/xml/ns/jeus”>
    <node>
        <name>@HOSTNAME@</name>
            <engine-container>
            <name>container1</name>
            <!-- engine container JVM option -->
            <command-option>
            . . .
                –Djeus.security.globalPassword=syspass
            </command-option>
            <engine-command>
                <type>servlet</type>
                <name>engine1</name>
                <startup-mode>xml</startup-mode>
            </engine-command>
       . . .
```

클러스터 내의 모든 VM 은 동일한 전역 시스템 패스워드를 가져야 한다는 것을 주의하기 바란다. 시스템 패스워드는 application client container 와 같은 client VM 에는 설정하지 않는다.

전역 시스템 패스워드를 별도로 설정하지 않았다면, 디폴트로 "globalpass"가 패스워드로 사용된다.

3.8 J2SE SecurityManger 설정하기

JEUS 에서 J2SE SecurityManger 를 사용하면, 플랫폼에 대한 부수적인 견고성을 얻을 수 있을지 몰라도, 퍼포먼스에 악영향을 끼친다. 일반적으로 J2SE SecurityManger 에서는, 모든 핵심 JEUS 코드뿐 아니라, JEUS 에 deploy 되어 있는 J2EE 어플리케이션 및 모듈이 완벽하게 신뢰받은 코드로 간주되므로, 보안 관리자를 굳이 사용하여 부하를 초래할 필요는 없다. 보안 관리자를 사용하지 않는 모드가 JEUS 에서 디폴트 모드이다.

그러나 때때로 J2SE SecurityManger 를 사용하여 코드 수준의 보안을 강화시켜 추가적인 견고성을 높이는 문제가 퍼포먼스 저하보다 중요하게 다루어지는 경우가 있다. 가령, 시스템 관리자가 미심쩍은 코드를 포함하고 있을 지도 모르는 J2EE 어플리케이션을 JEUS 에 deploy 해야 한다고 하자. 이 경우에 퍼포먼스 저하라는 비용을 감수하고서라도 코드 레벨의 보안를 강화시켜, 호스트를 보호하는 것이 더 중요하다고 판단되면, J2SE SecurityManger 를 작동시켜야 한다.

J2SE SecurityManger 를 JEUS 와 함께 동작 시키기 위한 절차는 다음과 같다.

JEUS 를 실행시키는 jeus.cmd 나 jeus 스크립트에 다음과 같은 속성을 정의해 준다.

-Djava.security.manager
-Djava.security.policy=%JEUS_HOME%\config\security\policy (윈도우기준)

그리고 policy 파일은 %JEUS_HOME%\config\security\policy 이며 내용은 다음과 같다.

<<*policy*>>

```
grant codeBase "file:${jeus.home}/lib/system/*" {
      permission java.security.AllPermission;
};
```

```
grant {
      permission java.net.SocketPermission "127.0.0.1:1024-",
"connect, accept, connect, listen, resolve";
      permission java.security.SecurityPermission
"runTrustedLogin", "read";
      permission java.security.SecurityPermission
"loginCodeSubject", "read";
      permission javax.management.MBeanPermission "*", "*";
};
```

주의: J2SE SecurityManger 는 JEUS 의 보안 시스템과는 완전히 별개의 것이다. JEUS 보안 시스템은 코드 수준의 보호(코드를 호출할 수 있는 Permission 설정) 가 아니라, 사용자 수준의 보호(누가 로그인했고, 해당 자원에 대한 Permission 이 있는가)를 다루고 있다. 둘 사이 유일한 접점은 JEUS 가 특별한 경우 J2SE SecurityManger 를 사용하여 코드 수준의 Permission 을 체크함으로써, 악의적인 Servlet 이나 EJB 코드로부터 JEUS 를 보호하기도 한다는 점이다.

3.9 JACC Provider 설정하기

JEUS 에서 JACC 1.0 설정에 대한 이슈는 10 장에서 충분히 설명할 것이다. 더 자세한 내용은 10 장을 참고하기 바란다.

3.10 결론

본 장에서는 JEUS 에서 핵심적인 보안 설정을 하는 방법을 알아 보았다. 즉 다음과 같은 설정 방법들을 알아보았다. 보안 도메인 설정법, 각 도메인에서 서비스를 설정하는 방법, subjects.xml 에 Subject 설정하는 방법, policies.xml 에 Policy 를 설정하는 방법, 추가적인 항목 설정 방법, JEUS 에서 J2SE SecurityManger 를 설정하는 방법, 마지막으로 JEUS 에서 JACC1.0 서비스를 설정하는 방법을 설명하였다.

본 장에서 설명한 서비스 이외의 서비스를 도메인에 설정하는 방법은, 부록 F 를 참고하기 바란다.

다음 장에서는 JEUS 에 deploy 된 J2EE 어플리케이션과 모듈에서 어떻게 보안을 설정하는지 알아보겠다.

4 어플리케이션과 모듈에서 보안 설정

4.1 소개

본 장에서는 J2EE 어플리케이션과 EJB 모듈, 웹 모듈에서 보안을 설정 하는 방법을 살펴 보겠다.

4.2 어플리케이션과 모듈 보안 개요

4.2.1 소개

본 절에서는 어플리케이션과 모듈에 보안을 설정하는 것과 관련된 가장 기본적인 이슈를 살펴 보겠다.

4.2.2 모듈 Deployment 대 Application Deployment

JEUS 에서 deploy 하는 방법은 크게 두가지로 나뉜다.

- EJB 모듈 (EJB .jar 파일) , 웹 모듈 (.war 파일)을 각각 독립적으로 deploy 하는 방식.
- 여러 개의 EJB, Web, connector 모듈(.rar)을 ear 로 압축하여, J2EE 어플리케이션(.ear)으로 deploy 하는 방식.

4.2.3 Role-to-Resource 매핑

Assembler 는 EJB 모듈이나 웹 모듈을 deploy 하기 전에, 해당 모듈에 role-to-resource 매핑을 설정해 두어야 한다. role-to-resource 매핑을 다른 말로 보안 제약(security constraint)이라고 하며, 논리적인 Role 에 모듈의 리소스를 맵핑하는 것을 말한다. 참고로 EJB 모듈에서 리소스란 EJB 의 메소드를 말하며, 웹 모듈에서 리소스란 Servlet URL 을 말한다. 이러한 매핑은 EJB 모듈에서는 META-INF\ejb-jar.xml 에, 웹 모듈에서는 WEB-INF\web.xml 에 설정한다. 아래는 EJB 모듈에 설정된 보안 제약 내용으로 META-INF\ejb-jar.xml 에 명시되어 있다.

<<ejb-jar.xml>>

```
<?xml version="1.0"?>
```

```
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>product</display-name>
    <enterprise-beans>
        <entity>
(A)         <ejb-name>product</ejb-name>
            . . .
(B)         <security-role-ref>
                <role-name>cust</role-name>
                <role-link>customer</role-link>
            </security-role-ref>
        </entity>
    </enterprise-beans>
    <assembly-descriptor>
        . . .
(C)     <security-role>
            <role-name>administrator</role-name>
        </security-role>
(D)     <security-role>
            <role-name>customer</role-name>
        </security-role>
(E}     <method-permission>
            <role-name>administrator</role-name>
            <method>
                <ejb-name>product</ejb-name>
                <method-intf>Remote</method-intf>
                <method-name>getSecretKey</method-name>
                <method-params>
                   <method-param>java.lang.Integer</method-param>
                </method-params>
            </method>
        </method-permission>
(F)    <method-permission>
            <unchecked/>
            <method>
                <ejb-name>product</ejb-name>
                <method-name>doSomeAdmin</method-name>
                <method-params>
                    <method-param>java.lang.String</method-param>
                </method-params>
```

```
            </method>
        </method-permission>
(G)     <method-permission>
            <role-name>customer</role-name>
            <method>
                <ejb-name>product</ejb-name>
                <method-name>test1</method-name>
            </method>
        </method-permission>
(H)     <exclude-list>
            <method>
                <ejb-name>product</ejb-name>
                <method-intf>Remote</method-intf>
                <method-name>getCustomerProfile</method-name>
                <method-params></method-params>
            </method>
        </exclude-list>
    </assembly-descriptor>
</ejb-jar>
```

위의 DD 파일이 말하고자 하는 보안 제약 사항은 다음과 같다.

A. "product"라 불리는 entity EJB 가 있다.

B. role-to-role reference 매핑이 선언되어 있다. 실제 선언된 Role 인 "customer"는 "cust"라는 Role Reference 로 참조될 수 있다. 이는 EJB 소스내에 "cust"라는 이름의 Role 은 실제로는 "customer" Role 에 해당한다는 것을 뜻한다.

C. 논리적 Role 인 "administrator"가 선언되어 있다.

D. 논리적 Role 인 "customer"가 선언되어 있다.

E. role-to-resource 매핑이 선언되어 있는데, 이는 "administrator" Role 이 리모트 EJB 인터페이스의 getSecreteKey 메소드를 호출할 수 있도록 허락한다. 이 매핑에 따라, 오직 "administrator" Role 에 속하는 Principal 만이 getSecreteKey 메소드를 호출할 수 있다.

F. doSomeAdmin 메소드는 Unchecked 로 선언되어 있다. 즉 누구나 Role 에 상관없이 이 메소드를 호출할 수 있다는 말이다.

G. "customer" Role 에 속하는 Principal 은 "product" EJB 의 "test1" 메소드를 호출할 수 있다.

H. getCustomerProfile 메소드는 Excluded 목록에 포함되어 있다. 즉 Role 에 상관없이, 어떤 Role 도 해당 메소드를 호출할 수 없다는 뜻이다.

주의: 웹 모듈에 대해서도 EJB 와 비슷한 설정이 적용된다. 단, EJB 에서 Role 이 EJB 메소드를 호출할 수 있느냐의 문제였다면, 웹 모듈에서는 Role 이 Servlet URL 에 접근할 수 있느냐의 문제를 다룬다. 웹 모듈 보안 설정에 대한 자세한 정보는 4.4 절을 참고하기 바란다.

지금까지 언급한 두가지 Role 즉 "administrator"와 "customer"는 순전히 논리적 개념으로, 실제 deploy 되는 환경을 의미하지 않는다. 따라서, 논리적 Role 을 실제 환경의 사용자나 사용자 그룹에 매핑하는 작업이 필요하다. 이 작업을 principal-to-role 매핑이라 하는데 자세한 정보는 다음 절에서 살펴 보겠다.

4.2.4 Principal-to-Role 매핑

보통 Deployer 가 J2EE 모듈 또는 어플리케이션에 정의된 Role 을 실제 환경의 사용자나 사용자 그룹에 매핑하는 작업을 한다. 여기서, Principal 을 논리적 Role 에 매핑하는 것은 어플리케이션 범위(scope)를 가진다는 점에 주의해야 한다.

예를 들어, 모듈 M1 과 M2 가 A1 이라는 어플리케이션에 포함되어 있다면, principal-to-role 매핑은 M1 과 M2 에서 공유된다. 더 나아가 A2 라는 별도의 어플리케이션은 A1 과는 완전히 다른 principal-to-role 매핑을 가진다. A1 과 A2 에서 동일한 이름의 Role 이 있다고 하더라도, 이들은 서로 다른 Role 로 인식되며, 공유되지 않는다. 이러한 범위와 관련된 개념을 [그림 15]에서 보여주고 있다.

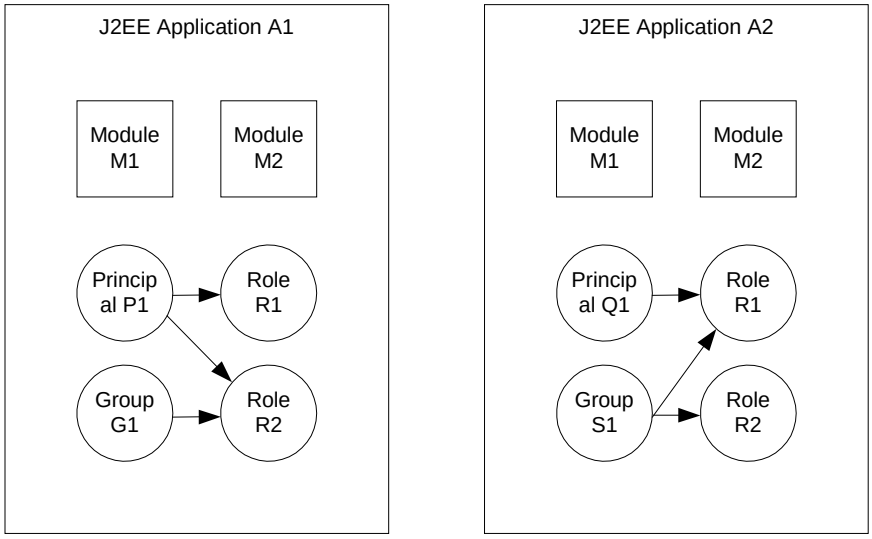

그림 15 Principal-to-Role 맵핑

[그림 15] 에서 보듯이 두 개의 서로 다른 어플리케이션인 A1 과 A2 가 서로 다른 principal-to-role 매핑을 가지고 있으며, 포함되어 있는 모듈(M1, M2)은 어플리케이션이 포함하고 있는 매핑을 공유한다.

JEUS 에서 principal-to-role 매핑은 JEUS DD 인 jeus-ejb-dd.xml(EJB 모듈) 과 jeus-web-dd.xml(웹 모듈), JEUSMain.xml(J2EE 어플리케이션)에 명시된다. 아래는 이전 절에 다루었던 ejb-jar.xml 에 대한 jeus-ejb-dd.xml 의 설정 예이다.

<<jeus-ejb-dd.xml>>

```
<?xml version="1.0"?>

<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
     <module-info>
(A)      <role-permission>
             <principal>johan</principal>
             <role>administrator</role>
             <classname>mypackage.MyRolePermission</classname>
         </role-permission>
(B)      <role-permission>
             <principal>peter</principal>
             <role>customer</role>
         </role-permission>
(C)      <role-permission>
             <role>roleA</role>
             </excluded>
```

```
        </role-permission>
(D)     <role-permission>
            <role>roleB</role>
            </unchecked>
        </role-permission>
(E)     <unspecified-method-permission>
            <role>administrator</role>
        </unspecified-method-permission>
    </module-info>
    <beanlist>
        . . .
    </beanlist>
</jeus-ejb-dd>
```

위의 예제는 다음과 같은 보안 정보를 담고 있다.

A. "johan"이라는 Principal(user) 에게 "administrator" Role 을 허가하는 Role Permission 이 선언되어 있다. 이는 "johan"이 "administrator" Role 의 모든 권한을 가지게 되었다는 것을 뜻한다. 예제에서 보듯이, 커스텀 Role Permission 은 <classname> 태그로 정의한다. 커스텀 Role Permission 에 대한 자세한 정보는 3 장을 참고하기 바란다.

B. "peter"라는 Principal(user)에게 "customer" Role 을 부여하는 Role-Permission 이 선언되어 있다. 이는 "peter"가 "customer" Role 이 가지는 모든 권한을 가지게 되었다는 뜻한다. classname 태그가 설정되어 있지 않았으므로, 디폴트 Role Permission 이 설정되어 있다.

C. "roleA" 라는 Role 은 모든 Principal 로부터 배제되어(excluded) 있다. 즉, 어떤 Principal 도 "roleA"에 포함될 수 없다.

D. "roleB"라는 Role 은 권한 부여 시스템에서 체크되지 않는다 (unchecked). 즉, 모든 Principal 은 자동으로 "roleB"에 포함된다.

E. unspecified-method-permission 은 "unspecified" EJB 메소드를 기본적으로 어떻게 다룰지 결정한다. "unspecified" EJB 메소드란 ejb-jar.xml 에서 <method-permission> 태그로 언급되지 않은 메소드를 말한다. unspecified-method-permission 은 3 가지 방식으로 처리할 수 있다. Role 에 맵핑 (예제에서는 "administrtor" Role 에 맵핑되었다)하거나, "excluded" (누구도 접근할 수 없는 것)로 취급하거나, "unchecked" (누구나 접근할 수 있는 것)로 취급하는 것이 그것이다. 마지막 "unchecked" 옵션이 디폴트이다.

주의: 웹 모듈 (jeus-web-dd.xml)의 설정은 unspecified-method-permission 설정이 없다는 것을 제외하고, EJB 모듈에서 설정과 동일하다.

4.2.5 결론

본 절에서 J2EE 어플리케이션, EJB 모듈, 웹 모듈의 가장 기본적인 보안 설정을 다루었다. 이상에서 보듯이 보안 설정은 크게 두 단계의 절차로 나뉜다.

1. role-to-resource 매핑을 각 모듈 마다 설정한다.

2. 각 어플리케이션에 대해 principal-to-role 매핑을 정의한다.

다음 절에서는 EJB 모듈에 보안 설정하는 방법을 자세히 알아 보겠다.

4.3 EJB 모듈에 보안 설정하기

4.3.1 소개

본 절에서 EJB 모듈에 보안 설정하는 방법을 자세하게 알아 보겠다.

논의는 크게 두가지 파트로 나눌 수 있다.

1. ejb-jar.xml 설정하기
2. jeus-ejb-dd.xml 설정하기

4.3.2 ejb-jar.xml 설정하기

ejb-jar.xml 에 다음과 같은 보안 요소를 설정할 수 있다.

1. role-to-role reference 매핑하기
2. security identity 설정하기
3. 논리적 Role 선언하기
4. EJB 메소드 Permission 설정하기

role-to-role reference 매핑은 **<security-role-ref>** 태그로 정의한다. 목적은 실제 논리적 Role 에 대한 참조를 설정하는 것이다. 그래서 EJB 소스에서는 실제 Role 대신 Role Reference 를 사용하게 된다. 다음 태그는 <security-role-ref> 태그 내에서 사용하는 태그이다.

- **role-name**: 실제 Role 에 대한 참조명으로, EJB 코드에는 실제 Role 명 대신 참조명을 사용한다.

- **role-link**: security-role 태그로 선언한 실제 Role 명으로, Role Reference 는 role-link 가 가리키는 Role 에 매핑된다.

모듈내에 있는 각 EJB 는 다른 EJB 로 원격지 호출을 할 때, 자신의 security-identity 를 전파한다. EJB 의 security identity 는 `<ejb-jar><enterprise-beans><type><security-identity>`태그로 선언하며, 다음과 같은 하위 태그를 가지고 있다.

- **use-caller-identity**: empty 태그로, 만일 설정되어 있으면, EJB 호출시 호출자 자신이 security identity 로 사용된다.

- **run-as**: 이 태그가 설정되어 있으면, <run-as>태그 내의 <role-name>에 설정된 Role 이 security identity 가 된다.

<assembly-descriptior>의 하위 태그인 **<security-role>** 태그는 해당 EJB 모듈에 적용되는 논리적 Role 을 선언하는 태그이다. ejb-jar.xml 에 언급되어 있는 모든 Role 명은 이 태그내에 선언되어 있어야 한다.

<assembly-descriptor> 의 하위 태그인 **<method-permission>**태그는 각 EJB 메소드에 보안 제약을 설정하는 태그이다. 하위 태그로는 다음과 같은 태그가 있다. :

- **role-name** (선택적): <security-role>태그에 선언되어 있는 논리적 Role 명으로 아래에 정의된 메소드에 접근할 수 있다.

- **unchecked** (선택적)**:** 값이 없는 empty 태그로, 설정되어 있으면 아래에 정의된 메소드는 권한 체크를 하지 않는다. (즉 Role 에 상관없이 누구나 메소드에 접근할 수 있다).

- **method** (1 개 이상)**:** role-name 또는 unchecked 태그가 적용되는 메소드로, 하위 태그는 다음과 같다.

  - **ejb-name**: 메소드를 포함하고 있는 EJB 명

  - **method-intf**: 메소드를 포함하고 있는 EJB 인터페이스 타입. 생략 되면, 로컬, 리모트 인터페이스 모두를 뜻하게 된다.

  - **method-name**: EJB 메소드 명.

- **method-params** (선택적,)**:** 0 개 이상의 **<method-param>** 태그를 가질 수 있다. method-param 태그는 EJB 메소드의 각 파라미터를 나타낸다. <method-params> 태그가 생략되어 있으면, 파라미터와 상관없이 메소드명만 동일하다면, 동일한 Permission 이 적용된다(EJB 에는 파라미터는 다르나, 메소드 명은 동일한 여러 개의 메소드가 있을 수 있다. 이 메소드들에 공통적으로 해당 Method Permission 이 적용된다는 뜻이다). <method-param> 태그 값은 패키지명을 포함한 완전한 Java 클래스명이거나, "int"와 같은 Java Primitive 타입이 될 수 있다. <method-params>를 empty 태그로 설정하면(<method-params/>) 매개변수가 하나도 없다는 것을 뜻한다.

- **exclude-list** (1 개)**:** 단 하나의 <exclude-list>가 올 수 있으며, Excluded 되는 EJB 메소드를 정의한다(즉 누구도 접근할 수 없는 메소드이다). 하위 태그로 1 개이상의 <method>태그를 가질 수 있다.

다음 예제는 위에서 설명한 태그를 이용하여 ejb-jar.xml 에 보안 설정을 한 것이다.

<<ejb-jar.xml>>

```
<?xml version="1.0"?>

<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee">
    <display-name>product</display-name>
    <enterprise-beans>
        <entity>
            <ejb-name>product</ejb-name>
            . . .
            <security-role-ref>
                <role-name>cust</role-name>
                <role-link>customer</role-link>
            </security-role-ref>
            <security-identity>
                <run-as>
                    <role-name>administrator</role-name>
                </run-as>
            </security-identity>
        </entity>
    </enterprise-beans>
```

```
<assembly-descriptor>
    . . .
    <security-role>
        <role-name>administrator</role-name>
    </security-role>
    <security-role>
        <role-name>customer</role-name>
    </security-role>
    <method-permission>
        <role-name>administrator</role-name>
        <method>
            <ejb-name>product</ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>getSecretKey</method-name>
            <method-params>
               <method-param>java.lang.Integer</method-param>
            </method-params>
        </method>
    </method-permission>
    <method-permission>
        <unchecked/>
        <method>
            <ejb-name>product</ejb-name>
            <method-name>doSomeAdmin</method-name>
            <method-params>
                <method-param>java.lang.String</method-param>
            </method-params>
        </method>
    </method-permission>
    <method-permission>
        <role-name>customer</role-name>
        <method>
            <ejb-name>product</ejb-name>
            <method-name>test1</method-name>
        </method>
    </method-permission>
    <exclude-list>
        <method>
            <ejb-name>product</ejb-name>
```

```
                <method-intf>Remote</method-intf>
                <method-name>getCustomerProfile</method-name>
                <method-params></method-params>
            </method>
        </exclude-list>
    </assembly-descriptor>
</ejb-jar>
```

예제에 대한 설명은 4.2.3 절을 참고하기 바란다.

태그 각각에 대한 자세한 설명은 EJB 스펙을 참고하기 바란다. 또한 JEUS 에서 EJB 를 deploy 하는 방법은 JEUS EJB 안내서를 참고하기 바란다.

4.3.3 jeus-ejb-dd.xml 설정하기

EJB 에서 대부분 role-to-method 매핑은 ejb-jar.xml 에 설정되어 있으나, ejb-jar.xml 에서 설정 되어 있지 않는 메소드를 어떻게 다룰지와 실제 principal-to-role 매핑을 어디서 선언할 지는 여전히 해결해야 할 문제이다. 이는 EJB .jar 파일의 META-INF 디렉토리에 존재하는 jeus-ejb-dd.xml 파일에서 다뤄진다.

jeus-ejb-dd.xml 에서 대부분 보안 관련된 설정의 <jeus-ejb-dd> 루트 태그 바로 아래에 있는 <module-info>태그 내에 설정된다. 이 태그의 자식 태그는 다음과 같다.

- 0 개이상의 **role-permission** 태그. 각 태그는 Role (principal-to-role 매핑)을 정의하고 있다. 자식 태그는 다음과 같다.
  - **principal**(0 개 이상): Role 에 매핑되는 Principal 명
  - **role** (필수적): 단 하나만 허락되며, 필수 항목으로 실제 논리적 Role 명이다. role permission 을 구현하는 Java 클래스의 생성자에 첫번째 파라미터로 넘겨진다. Role 명은 ejb-jar.xml 에 선언된 Role 명과 일치해야 한다.
  - **actions** (선택적): 선택항목으로, role permission 에 대해 부가적인 정보를 제공한다. 만약 설정되어 있다면, 액션 데이터는 role permission 을 구현하는 Java 클래스의 두 번째 파라미터로 넘겨진다.
  - **classname** (선택적): role permission 을 구현하는 클래스로, 패키지 명을 포함한 완전한 이름이어야 한다. 이 클래스는 java.security.Permission 클래스의 서브 클래스로 최소한 하나의 파라미

터(role 의 이름)를 받는 public 생성자를 가지고 있어야 한다. 생략되어 있다면, jeus.security.resource.RolePermission 이 사용된다.

- **Excluded** (선택적): 선택 사항이며 empty 태그로, 만약 설정되어 있다면 해당 Role 은 excluded 된 것으로 취급한다(누구도 Role permission 에서 언급한 Role 에 접근할 수 없다). 이 태그는 <principal>, <unchecked> 태그보다 우선순위가 높다.

- **unchecked** (선택적)**::** 선택항목이며 empty 태그로, 만약 설정되어 있다면, 해당 Role 은 unchecked 된 것으로 취급한다(누구나 Role permission 에서 언급한 Role 에 접근할 수 있다). 이 태그는 <principal> 태그보다 우선 순위가 높다.

ejb-jar.xml 에 명시되어 있지 않는 EJB 메소드들을 어떻게 다루어야 할까? 여기에는 3 가지 방법이 있다. 이는 <module-info> 아래의 **<unspecified-method-permission>** 태그내에 정의된다. 가능한 자식 태그는 다음과 같다.

- **Role** (선택적): 설정되어 있다면, 모든 설정되지 않는 메소드는 이 Role 에 매핑된다.

- **excluded** (선택적): 설정되어 있다면, 모든 설정되지 않는 메소드는 excluded 취급된다(누구도 접근할 수 없다). 이 태그는 "role"이나 "unchecked"태그보다 우선 순위가 높다.

- **unchecked**(선택적): 설정되어 있다면, 모든 설정되지 않은 메소드는 unchecked 로 취급된다(누구나 접근할 수 있다). 이 태그는 "role" 태그보다 우선 순위가 높다.

마지막으로, run-as identity 를 사용하는 EJB 에 대해 Principal 을 설정하는 방법을 알아보자. 이는 `<jeus-bean><run-as-identity><principal-name>` 태그 안에 주어진 `Principal` 명이 사용된다. `Principal` 은 `ejb-jar.xml` 에서 `<security-identity><run-as><role-name>`태그내에 명시된 `role` 내에 포함되어 있어야 한다.

예제는 다음과 같다:

<<*jeus-ejb-dd.xml*>>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns=“http://www.tmaxsoft.com/xml/ns/jeus”>
     <module-info>
        <role-permission>
```

```
            <principal>johan</principal>
            <role>administrator</role>
            <classname>mypackage.MyRolePermission</classname>
        </role-permission>
        <role-permission>
            <principal>peter</principal>
            <role>customer</role>
        </role-permission>
        <role-permission>
            <role>roleA</role>
            </excluded>
        </role-permission>
        <role-permission>
            <role>roleB</role>
            </unchecked>
        </role-permission>
        <unspecified-method-permission>
            <role>administrator</role>
        </unspecified-method-permission>
    </module-info>
    <beanlist>
        <jeus-bean>
            . . .
            <run-as-identity>
                <principal-name>johan</principal-name>
            </run-as-identity>
        </jeus-bean>
        . . .
    </beanlist>
</jeus-ejb-dd>
```

예제에 대한 자세한 설명은 4.2.3 절을 참고하기 바란다.

각 태그에 대한 자세한 정보는 JEUS EJB 가이드를 참고하기 바란다.

4.3.4 결론

지금까지 ejb-jra.xml, jeus-ejb-dd.xml 에 principal-to-role 과 role-to-method 매핑을 선언하는 방법을 알아 보았다.

본 절에서 다루지 않은 CSI 와 같은 보안 설정들은 JEUS EJB 가이드를 참고하기 바란다.

4.4 웹 모듈에서 보안 설정하기

4.4.1 소개

본 절에서 WEB 모듈 (Servlet 모듈)에 보안 설정하는 방법을 살펴 보겠다.

내용은 크게 두 가지 파트로 구성된다.

1. web.xml 설정하기
2. jeus-web-dd.xml 설정하기

4.4.2 web.xml 설정하기

web.xml 은 웹 아카이브 (war 파일)에 대한 메인 DD 파일이다. 표준 J2EE DD 파일로, war 아카이브 내에 WEB-INF 디렉토리내에 있다.

web.xml 에서 설정할 수 있는 보안 요소는 다음과 같다.

1. Run-as identity.
2. Role-to-role reference 매핑
3. Servlet URL 접근 Permissions (security constraints, 보안제약).
4. 논리적 Role 선언

웹 모듈에 포함되어 있는 Servlet 은 EJB 로 원격 호출을 할 때, 자신의 security identity 를 전파한다. Servlet 의 security identity 는 `<web-app><servlet><run-as>` 태그내에 설정되고, 다음과 같은 하위 태그를 가진다.

- **role-name**(선택적): Servlet 을 실행하는 Role, 생략하면 호출자의 identity 가 사용된다.

role-to-role reference 매핑은 **`<security-role-ref>`** 태그로 명시하며, Role Reference 를 실제 논리적 Role 에 매핑하는 것이 목적이다. Role Reference 는 Servlet 코드에서 Role 을 나타낼 때 사용된다. web.xml 에서 **`<security-role-`**

ref> 태그는 <servlet> 태그 내에 설정되며, 다음과 같은 하위 태그를 가진다.

- **role-name**: Servlet 코드에 사용되는 Role 명.
- **role-link**: <security-role> 태그로 정의한 실제 Role 명. <role-name>에 설정된 Role Reference 를 실제 Role 에 매핑한다.

Servlet URL 에 접근을 제한 하려는 목적으로 <web-app> 태그 바로 아래에 있는 **<security-constraint>** 태그를 사용한다. <security-constraint> 는 다음과 같은 하위 태그를 가진다.

- **web-resource-collection** (1 개 이상)**:** 접근 제한이 설정되는 웹 리소스의 목록을 나타낸다.
  - **web-resource-name**: 웹 리소스 명
  - **url-patterns** (0 개 이상): 웹 리소스를 나타내는 URL(contextroot 에 상대경로로 지정) 패턴이다. URL 패턴의 예로는 "/mywebapp/*" (mywebapp 가 포함된 모든 URL), "/something" (정확한 URL), "*.jsp" (jsp 로 끝나는 모든 리소스) 가 있다.
  - **http-method** (0 개 이상)**:** 웹 리소스에 적용되는 HTTP 메소드를 나타낸다. "GET", "POST", "PUT"등이 올 수 있다.
- **auth-constraint** (선택적)**:** <security-constraint> 태그 내에 정의된 웹 리소스에 접근할 수 있는 Role 을 설정한다. 하위 태그로는 0 개 이상의 **role-name** 태그가 오는데, 각각은 접근이 허락된 Role 을 나타낸다. 만약 Role 명이 설정되어 있지 않고, empty 태그로 (<auth-constraint/>) 설정되어 있다면, 아무도 해당 웹 리소스에 접근할 수 없다는 뜻이다(이는 웹 리소스가 Excluded 된다는 것과 동일하다). 만약 <auth-constraint> 태그가 아예 생략되어 있다면, 이는 웹 리소스가 Unchecked 된다는 것으로, Role 에 상관없이 누구나 웹 리소스에 접근할 수 있다는 뜻이다.
- **user-data-constraint** (선택적)**:** 웹 리소스에 맺어지는 커넥션을 "transport guarantee"(전송 보장) 할 것인가를 선언하는 태그다. 하위 태그로 <**transport-guarantee>** 태그를 가지며, 웹 리소스로의 커넥션을 보장하는 레벨을 나타낸다. 가능한 값으로는 "NONE", "INTEGRAL", "CONFIDENTIAL"이 있다. "NONE"은 커넥션을 보장하지 못한다는 뜻이고, "INTEGRAL"은 메시지의 무결성(즉 보내

진 메시지는 변경되지 않은 원본이다)을 보장하는 커넥션이라는 뜻이고, 마지막으로 "CONFIDENTIAL"은 메시지가 도청되는 것을 방지하기 위해 암호화 되어 보내진다는 뜻이다.

마지막으로 <security-constraint> 태그 다음에 논리적 Role 을 나타내는 <security-role>태그가 온다. 각 태그는 <role-name> 태그를 가지고 있는데, 이미 DD 에 언급되었던 논리적 Role 명을 선언하는 부분이다.

아래에서 web.xml 에 보안 설정을 한 예제를 보여준다.

<<*web.xml*>>

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        . . .
(A)     <run-as>
            <role-name>R1</role-name>
        </run-as>
(B)     <security-role-ref>
            <role-name>adminRef</role-name>
            <role-link>R1</role-link>
        </security-role-ref>
        . . .
    </servlet>
(C) <security-constraint>
       <web-resource-collection>
            <web-resource-name>A</web-resource-name>
            <url-pattern>/a/*</url-pattern>
            <url-pattern>/b/*</url-pattern>
            <url-pattern>/a</url-pattern>
            <url-pattern>/b</url-pattern>
            <http-method>DELETE</http-method>
            <http-method>PUT</http-method>
        </web-resource-collection>
        <web-resource-collection>
            <web-resource-name>B</web-resource-name>
            <url-pattern>*.asp</url-pattern>
        </web-resource-collection>
        <auth-constraint/>
```

```
    </security-constraint>
(D) <security-constraint>
        <web-resource-collection>
            <web-resource-name>C</web-resource-name>
            <url-pattern>/a/*</url-pattern>
            <url-pattern>/b/*</url-pattern>
            <http-method>GET</http-method>
        </web-resource-collection>
        <web-resource-collection>
            <web-resource-name>D</web-resource-name>
            <url-pattern>/b/*</url-pattern>
            <http-method>POST</http-method>
        </web-resource-collection>

        <auth-constraint>
            <role-name>R1</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>
                CONFIDENTIAL
            </transport-guarantee>
        </user-data-constraint>
    </security-constraint>
(E) <security-role>
        <role-name>R1</role-name>
    </security-role>
    <security-role>
        <role-name>R2</role-name>
    </security-role>
    <security-role>
        <role-name>R3</role-name>
    </security-role>
</web-app>
```

예제를 설명하자면:

A. run-as-identity 가 Role "R1"으로 설정되어 있다는 말은, Servlet 의 security identity 가 "R1"에 속한 Principal 이어야 한다는 뜻이다. 실제 Principal 은 JEUS DD 파일에서 설정된다.

B. security-role-ref 태그는 실제 Role 인 "R1"을 Role Reference 인 "adminRef"로 매핑해 놓았는데, 실제 Servlet 소스 내에서는 Role 대신 Role Reference 인 "adminRef"를 사용한다.

C. 첫번째 security constraint 설정은, /a, /b, /a/*, /b/* URL 패턴을 가지고 있으면서 HTTP 메소드가 DELETE, PUT 인 URL 과 *.asp 로 끝나는 URL 은 누구도 접근할 수 없다는 것을 말하고 있다. (<auth-constraint/>로 설정되어 있기 때문이다).

D. 두 번째 security constraint 설정은 /a/*, /b/* 패턴을 가지고 있으면서 GET 방식인 URL 과 /b/* 패턴을 가지고 있으면서 POST 방식인 URL 은, 오직 "R1" Role 에 포함된 Principal 만 접근할 수 있다는 것을 말하고 있다. 그리고 해당 커넥션은 CONFIDENTIAL 을 보장한다.

E. "R1", "R2", "R3" 세개의 논리적 Role 을 선언하고 있다.

주의: 웹 리소스 (URL 패턴 + HTTP 메소드) 가 web.xml 에 별도로 설정되어 있지 않으면, 해당 리소스는 Role 에 상관없이 누구나 접근할 수 있다는 뜻이다. 이는 J2EE 에서 웹어플리케이션 접근에 대한 디폴트 값이다.

web.xml 에 나타나는 모든 태그들에 대한 자세한 정보는 Servlet 스펙을 참고하기 바란다. 또한 JEUS 에서 Servlet 을 deploy 하는 것에 대한 추가적인 정보는 JEUS Web Container 안내서를 참고하기 바란다.

4.4.3 jeus-web-dd.xml 설정하기

이전 절에서 web.xml 에서 웹 모듈에 대한 role-to-web 리소스를 정의하는 방법을 알아 보았다.

지금부터 웹 모듈에 대한 JEUS DD 파일인 jeus-web-dd.xml 에서 보안을 설정하는 방법에 대해 알아 보자.

jeus-web-dd.xml 에서 principal-to-role 매핑을 정의하기 위해, `<jeus-web-dd><context><role-mapping>` 태그를 사용한다. 아래는 **`<role-mapping>`** 태그의 하위 태그들이다.

- **role-permission** (0 개 이상): Role 에 대한 Permission 을 (보통 principal-to-role 매핑) 설정한다. 다음과 같은 하위 태그를 가진다.
  - **principal** (0 개 이상)**:** Role 에 매핑되는 Principal 명이다.

- **role** (1 개, 필수적)**:** 논리적 Role 명이다. Role 명은 Role Permission 을 구현하는 Java 클래스 생성자에 첫번째 파라미터로 전달된다. Role 명은 web.xml 에 주어진 논리적 Role 명과 일치해야 한다.

- **actions** (선택적)**:** Role Permission 에 대한 추가적인 데이터를 전달한다. 설정되어 있다면, Role Permission 을 구현하는 Java 클래스 생성자의 두 번째 파라미터로 전달된다.

- **classname** (선택적)**:** Role Permission 을 구현하는 Java 클래스명이다. 이 클래스는 java.security.Permission 의 구현 클래스로 최소한 하나의 파라미터 (String rolename)를 넘겨받는 public 생성자를 가지고 있어야 한다. 생략되면, 디폴트로 “jeus.security.resource.RolePermission” 클래스가 사용된다.

- **excluded** (선택적)**:** empty 태그로, 설정되어 있으면 Role 은 Excluded 된 것으로 취급한다(즉 어떤 Principal 도 해당 Role 을 부여 받을 수 없다). 이 태그는 "principal", "unchecked" 태그보다 우선 순위가 높다.

- **unchecked** (선택적)**:** empty 태그로, 설정되어 있으면 Role 은 Unchecked 된 것으로 취급한다(즉 Principal 에 상관 없이 누구나 Role 을 부여 받을 수 있다). 이 태그는 "principal" 태그보다 우선 순위가 높다.

Servlet 에서 run-as-principal 을 설정하기 위해, <role-mapping> 태그 다음에 <run-as><principal-name> 태그를 추가한다. <principal-name> 태그 값은 web.xml 에서 <run-as><role-name>에 선언된 Role 에 속하는 Principal 이어야 한다.

주의: 현재로는 web.xml 에서 언급되어 있지 않는 Servlet URL 을 어떻게 다룰지 정의하는 태그가 jeus-web-dd.xml 에 없다. Servlet 스펙에서에서는 모든 정의되지 않은 URL 을 항상 Unchecked 로 취급한다(즉 누구나 접근할 수 있다).

예제:

```
<?xml version="1.0"?>

<jeus-web-dd xmlns=“http://www.tmaxsoft.com/xml/ns/jeus”>

        <context-path>/tutorial</context-path>
```

```
        <docbase>Tutorial</docbase>
        . . .
        <role-mapping>
(A)         <role-permission>
                <principal>johan</principal>
                <principal>peter</principal>
                <principal>customerGroup</principal>
                <role>R1</role>
            </role-permission>
(B)         <role-permission>
                <role>R2</role>
                </excluded>
            </role-permission>
(C)         <role-permission>
                <role>R3</role>
                </unchecked>
            </role-permission>
(D)         <role-permission>
                <principal>tellerGroup</principal>
                <role>R4</role>
                <actions>09:00-17:00</actions>
                <classname>
             jeus.security.resource.TimeConstrainedRolePermission
                </classname>
            </role-permission>
        </role-mapping>
(E)     <run-as>
            <principal-name>johan</principal-name>
        </run-as>
        . . .

</jeus-web-dd>
```

위의 예제는 다음을 말하고 있다.

A. "johan", "peter", "customerGroup" Principal 은 "R1" Role 에 속한다.

B. "R2" Role 은 Excluded 되어 있다. 즉 아무도 "R2" Role 을 부여 받을 수 없다.

C. "R3" Role 은 Unchecked 되어 있다. 즉 누구나 "R3" Role 을 부여 받을 수 있다.

D. "tellerGroup" Principal 은 오전 9 시부터 오후 5 시까지 Role "R4"를 부여 받는다(jeus.security.resource.TimeConstrainedRolePermission 클래스의 implies()메소드에서 구현된다).

E. run-as-principal 로 "johan"이 사용되었다(즉 Servlet 에서 EJB 를 호출할 때 "johan"이 security identity 로 넘겨 진다). deployer 는 "johan"이 web.xml 에 정의된 "R1" Role 에 속하는지 반드시 확인해야 한다.

4.4.4 결론

지금까지 웹 모듈에서 보안을 설정하는 방법을 알아 보았다. 이는 기본적으로 두단계로 구성되어 있다.

1.web.xml 에서 role-to-web resource 매핑 설정하기
2.jeus-web-dd.xml 에서 principal-to-role 매핑 설정하기

다음 장에서 J2EE 어플리케이션 (ear 파일)에서 보안을 설정하는 방법을 살펴보겠다.

4.5 J2EE 어플리케이션에 보안 설정하기

4.5.1 소개

본 절에서 J2EE 어플리케이션(ear 파일)의 보안을 설정하는 방법을 살펴 보겠다.

4.5.2 application.xml 설정하기

J2EE 어플리케이션은 .ear 확장자를 가진 아카이브 파일이다. ear 아카이브는 EJB 모듈, 웹 모듈, Connector 모듈, 그리고 모듈에서 참조하고 있는 클래스들로 구성되어 있다. "META-INF" 디렉토리는 application.xml 이란 설정 파일을 포함하고 있는데, 전체 어플리케이션에 대한 여러 가지 정보를 제공한다.

본 절에서는 application.xml 에서 보안과 관련된 부분을 알아본다. 기본적으로 application.xml 에는 보안과 관련된 설정은 논리적 Role 을 설정하는 단 한 가지 밖에 없다. Role 선언은 ear 파일 내에 있는 모든 EJB 모듈과 웹 모듈에 공통으로 적용되며 어플리케이션 범위(scope)를 가진다.

예제 :

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="1.4" . . .>
    <description>Application description</description>
    <display-name>myApp</display-name>
    <module>
        <web>
            <web-uri>myWebApp.war</web-uri>
            <context-root>myWebApp</context-root>
        </web>
    </module>
    <module>
        <ejb>myEjbApp.jar</ejb>
    </module>
(A) <security-role>
        <role-name>Administrator</role-name>
    </security-role>
(B) <security-role>
        <role-name>Customer</role-name>
    </security-role>
</application>
```

위의 예제에서 두 가지 role - "Administrator" (A), "Customer" (B) - 이 선언되었다. 두 가지 Role 은 role-to-resource 매핑을 정의하기 위해 EJB 와 웹 모듈의 DD 파일에서 사용된다 (EJB 와 웹 모듈 보안에 대해서는 이전 절에서 이미 설명되었다).

어플리케이션을 JEUS 서버에 deploy 하기 전에, 논리적 Role 이 실제 Principal 에 매핑되었는지 확인해야 한다. 이 매핑은 어플리케이션의 경우 JEUSMain.xml 에, EJB 모듈의 경우 jeus-ejb-dd.xml 에, 웹 모듈의 경우 jeus-web-dd.xml 에 설정된다. jeus-ejb-dd.xml 과 jeus-web-dd.xml 에서 principal-to-role 설정은 이미 설명하였다. 따라서, 다음 절에서 JEUSMain.xml 에서 principal-to-role 매핑을 설정하는 방법을 알아 보겠다.

참고: 어플리케이션 전체에서 Role 과 principal-to-role 매핑을 공유하기 때문에, J2EE 어플리케이션에 포함되어 있는 모든 모듈을 잘 알고 있어야 한다. 가령, application.xml 에 "Administrator"라는 Role 을 설정해 놓고, JEUSMain.xml 에서 "peter"라는 Principal 을 할당했다고 하자. 이 Role 매핑은 어플리케이션에 포함된 모든 EJB 모듈과 웹 모듈에서 적용된다.

비슷하게 Role 과 principal-to-role 매핑을 특정 EJB 의 jeus-ejb-dd.xml 에만 정의해 두었다더라도, 설정된 Role 과 principal-to-role 매핑은 .ear 에 포함된 모든 모듈에서 여전히 공유될 수 있다.

J2EE 어플리케이션에서 모든 principal-to-role 매핑은 그것이 JEUSMain.xml 에 설정되어 있든, 특정 jeus-ejb-dd.xml 이나, jeus-web-dd.xml 에 설정되어 있든 상관없이 application scope 를 가진다는 점을 명심하기 바란다.

4.5.3 JEUSMain.xml 설정하기

어플리케이션 레벨에서 principal-to-role 매핑은 JEUSMain.xml 에 설정한다.

principal-to-role 매핑을 정의하기 위해, JEUSMain.xml 의 `<jeus-system><application>` 태그 아래에 다음 태그들을 추가한다.

- **role-permission** (선택적): 각 role-permission 태그는 principal-to-role 매핑을 정의하며, 선택적으로 Excluded, Unchecked Role 을 설정할 수 있다. 이 태그는 jeus-web-dd.xml, jeus-ejb-dd.xml, policies.xml 과 완전 동일하게 동작한다. 다음의 하위 태그를 가진다.
  - **principal** (0 개 이상)**:** Role 에 매핑되는 Principal 명이다.
  - **role**(1 개, 필수적)**:** Role 이름을 나타낸다. 이는 Role Permission 을 구현하는 Java 클래스 생성자에 첫번째 파라미터로 넘겨진다. Role 명은 application.xml 에 선언된 논리적 Role 명과 일치해야 한다.
  - **actions** (선택적)**:** Role Permission 에 부가적인 정보를 제공한다. 설정되어 있다면, Role Permission 을 구현하는 Java 클래스 생성자에 두 번째 파라미터로 넘겨진다.
  - **classname** (선택적)**:** Role Permission 을 구현하고 있는 Java 클래스명이다. 이 클래스는 java.security.Permission 클래스의 서브 클래스로 최소한 하나의 파라미터 (String rolename)를 넘겨 받는 public 생성자가 있어야 한다. 생략되어 있다면, 디폴트로 “jeus.security.resource.RolePermission”가 사용된다.
  - **excluded** (선택적)**:** empty 태그로, 설정되어 있으면, Role 은 Excluded 로 취급된다. (즉 아무도 해당 Role 을 부여받을 수 없다). 이 태그는 "principal"이나 "unchecked"태그보다 우선 순위가 높다.

- **unchecked** (선택적)**:** empty 태그로, 설정되어 있으면, Role 이 Unchecked 로 취급된다(즉 누구나 해당 Role 을 부여 받을 수 있다). 이 태그는 "principal"태그보다 우선 순위가 높다.

또한, <application> 태그 아래에 해당 어플리케이션이 deploy 되는 보안 도메인을 설정해 놓아야 한다. 디폴트로 "SYSTEM_DOMAIN"이 사용되지만, 특정 어플리케이션이 특별한 보안 서비스를 필요로 한다면, 해당 보안 서비스를 포함하고 있는 새로운 도메인을 만들어서, deploy 해야 한다. **<security-domain-name>**태그에 도메인명을 설정한다.

예제 :

<<JEUSMain.xml>>

```
<jeus-system>
    . . .
    <application>
        . . .
(A)     <security-domain-name>MY_DOMAIN</security-domain-name>
(B)     <role-permission>
           <principal>peter</principal>
           <role>Administrator</role>
        </role-permission>
(C)     <role-permission>
           <role>Customer</role>
           </unchecked>
        </role-permission>
        . . .
    </application>
</jeus-system>
```

위의 예제에서, J2EE 어플리케이션 보안을 다음과 같이 설정했다.

A. "MY_DOMAIN"이라는 보안 도메인에 어플리케이션을 deploy 했다.

B. "peter"라는 Principal 은 "Administrator" Role 에 매핑되어 있으며, 디폴트 RolePermission 을 사용한다.

C. 모든 Principal 에 "Customer" Role 을 부여한다.(Role 이 "unchecked" 되었기 때문이다).

보안 도메인을 설정하는 방법은 3.3 절을 참고하기 바란다. 커스텀 Permission 을 개발하고 설정하는 방법은 3.6 절을 참고하기 바란다. JEUS 에 J2EE 어플리케이션을 deploy 하는 방법은 JEUS Server 안내서를 참고하기 바란다.

4.5.4 결론

본 절에서는 J2EE 어플리케이션에 보안 정보를 제공하기 위해, application.xml 과 JEUSMain.xml 에 보안을 설정하는 방법을 알아 보았다. 정리하자면, application.xml 에는 논리적 Role 을 선언하고, JEUSMain.xml 에 deploy 할 보안 도메인과 principal-to-role 매핑을 설정한다.

4.6 결론

지금까지 EJB 모듈, 웹 모듈, J2EE 어플리케이션에서 보안을 설정하는 방법을 알아 보았다.

다음 장에서 JEUS 보안 시스템을 런타임에 어떻게 운영하는지 알아 보겠다.

5 보안 시스템 운영

5.1 소개

본 장에서는 securityadmin 명령어 라인 툴을 이용해서 런타임에 보안 시스템을 운영하는 방법에 대해 설명한다.

구체적으로, Subject 를 추가하는 방법, Subject 에 대한 Role Permission 을 추가하는 방법, Role 에 대해 Resource Permission 을 추가하는 방법, 마지막으로 위의 설정들을 취소시키는 방법에 대해 알아 보겠다.

또한 보안 정보가 전혀 설정되어 있지 않은 새로 생성된 보안 도메인을 어떻게 적용하는지도 알아 본다.

5.2 보안 시스템에 보안 정보 추가하기

먼저 JEUS 서버 가이드에 설명된 대로 JEUS Server 를 부팅한다.

JEUS 가 작동중인 머신에 명령어 프롬프트 창을 열고서,"securityadmin"이라고 입력한다 (JEUS_HOME\bin 디렉토리에 있다).

```
>securityadmin
Security Administration Console Version 1.0
?
```

“?”의 의미는 securityadmin 이 remote 에 있는 JEUS Security 서비스와 연계하여 동작할 준비가 되었다는 뜻이다.

명령어에 대한 도움말을 보기 위해서, help 다음에 명령어를 입력한다.

```
? help

Basic commands
    help.................: print this help message
    help <command name>..: print help about command
    asserttrue <command>.: assert that command does not fail
```

```
    assertfalse <command>: assert that command fails
    echo <text>.........: print text
    #...................: ignore line (comment)
    exit................: quit the tool
Note: To include spaces in a string, surround string with ""
Example: "Hello World!" will be treated as one string, not two

Login & logout commands
    login
    login2
(etc. . .)
```

특정 명령어에 대한 도움말을 보기 위해서, help 다음에 명령어를 입력한다.

```
? help login
login
    Log in the specified user using a basic password-based
    mechanism
    Arguments:
     -username <the username>
     -password <the password>
    [-domain] <the security domain name>
    Legend: [ ] = optional arg, <xyz> = replace xyz w. real value
    Example: somecommand -arg1 value1 -arg2 value2
?
```

administrator 계정으로 툴에 로그인해 본다.

```
? login -username johan -password johanpw
User johan successfully logged in
?
```

Subject 를 시스템에 추가해 본다.

```
? addsubject -username peter
Added Subject peter
?
```

"peter"에 대한 패스워드를 설정해 본다.

```
? setpassword -username peter -password peterpw
Password set for peter
```

```
?
```

Subject 에 대한 정보를 가져와 본다.

```
? getsubject -username peter
[SUBJECT]
Description: No description
Domain: SYSTEM_DOMAIN
Main principal: Principal peter
Principals: [Principal peter]
Public credentials: []
Private credentials: [jeus.security.resource.Password@0]
Credential factories: [{password=cGV0ZXJwdw==}]
?
```

현재 보안 도메인에 설정되어 있는 Subject 의 목록을 가져와 본다.

```
? getsubjectnames
1. javajoe
2. johan
3. peter
4. j2ee_vi
5. j2ee
6. jeus
Got 6 Subject names
?
```

"peter"에 대한 role permission(principal-to-role 매핑) 을 추가해 본다.

```
? assignrole -principal peter -role customer
Assigned role "customer" to Principal "peter"
?
```

To assign a resource to the role “customer”, type:

"customer" role 에 리소스를 할당해 본다.

```
? assignresource -role customer -resource jndi
Assigned resource "jndi" to role "customer"
?
```

"peter"로 로그인 해 본다.

```
? login -username peter -password peterpw
User peter successfully logged in
?
```

"jndi" 리소스에 "peter"가 접근할 수 있는지 확인해 본다.

```
? authorizeresource -resource jndi
Resource granted for current Subject
?
```

"peter"를 로그아웃하면, "johan"이 다시 활성화 된다.

```
? logout
Subject peter logged out
?
```

로그인된 Subject 에 대한 정보를 가져온다.

```
? currentsubject
[SUBJECT]
Description: No description
Domain: SYSTEM_DOMAIN
Main principal: Principal johan
Principals: [Principal johan]
Public credentials: []
Private credentials: [jeus.security.resource.Password@aaa5c537]
Credential factories: []
?
```

현재 Subjects 메모리의 정보를 저장한다.

```
? savesubject
subjects saved
```

현재 Policies 메모리의 정보를 저장한다.

```
? savepolicy
policies saved
```

5.3 보안 시스템에서 보안 정보 제거하기

JEUS Server 를 시작하고 "securityadmin" 툴을 시작한 다음, 이전 절에서 설명한 대로 "administrator"로 로그인한다.

Resource Permission 을 제거해 본다.

```
? unassignresource -role customer -resource jndi
Unassigned resource "jndi" from role "customer"
?
```

Role Permission 을 제거해 본다.

```
? unassignrole -principal peter -role customer
Unassigned role "customer" from Principal "peter"
?
```

Subject 를 제거해 본다.

```
? removesubject -username peter
Removed Subject peter
?
```

툴을 종료한다.

```
? exit
Bye!
Executed 15 commands during this session.
>
```

5.4 새로 생성된 도메인 적용하기

간혹 Subject 도 Policy 도 설정되어 있지 않는 새로운 보안 도메인을 만들 경우가 있다. 그럼에도 여전히 securityadmin 툴로 해당 도메인에 로그인해서 보안 관련 작업을 해야 한다면, 어떻게 해야 할까? 그런 경우에는 "code Subject"라는 특별한 Subject 로 로그인하면 된다. "code Subject"는 Security Repository 를 변경하는 것을 포함한 모든 권한을 가지고 있는 특별한 Subject 이다.

새로운 도메인을 적용하기 위해 다음의 방법을 추천한다.

우선, securityadmin 스크립트 파일을 열어서 다음 시스템 속성값을 설정한다.

```
-Djeus.security.globalPassword=<global system password>
```

이때 전역 시스템 패스워드는 서버 쪽에 설정된 것과 일치해야 한다(자세한 정보는 3.7 절을 참고하기 바란다).

JEUS Manager 가 작동 중인 머신의 securityadmin 툴을 띄우고, 적용하기 원하는 새로운 도메인 명을 지정한다.

```
>securityadmin -domain TEST_DOMAIN
Security Administration Console Version 1.0
?
```

작업을 하기 원하는 도메인에 "code Subject"로 로그인 해본다.

```
? logincode -domain TEST_DOMAIN
Code Subject for domain TEST_DOMAIN logged in
?
```

새로운 administrator 계정인 "root"를 추가해 본다.

```
? addsubject -username root
Added Subject root
?
```

주의: 위 명령어는 로컬 전역 시스템 패스워드가 서버쪽에 설정되어 있는 전역 시스템 변수와 일치할 때만 작동한다.

"root"에 대한 패스워드를 설정해 본다.

```
? setpassword -username root -password sasquatch
Password set for root
?
```

"root"에 "RootAdministrator"라는 Role Permission 을 추가해 본다.

```
? assignrole -principal root -role RootAdministrator
Assigned role "RootAdministrator" to Principal "root"
?
```

"RootAdministrator" Role 에 "all-resource-permission"을 추가해 본다(즉 "root"는 리소스에 대한 모든 permission 을 가지게 된다).

```
? assignresource -role RootAdministrator -resource * -actions *
Assigned resource "*" to role "RootAdministrator"
```

```
?
```

"root"는 모든 Permission 을 가지고 있으므로, 이 계정으로 보안 시스템을 포함한 JEUS 의 나머지 시스템도 관리할 수 있다.

"code Subject"에서 로그아웃 해 본다.

```
? logout
Subject code logged out
?
```

"root"로 로그인 해본다.

```
? login -username root -password sasquatch
User root successfully logged in
?
```

이제 어떤 보안 관련 명령어도 실행할 수 있을 것이다.

참고: 툴을 종료한다음, securityadmin 스크립트에서 `jeus.security.globalPassword` 를 제거하기 바란다. 그리고 전역 시스템 패스워드는 항상 기밀성을 유지해야 한다.

5.5 결론

지금까지 securityadmin 툴로 보안과 관련된 사항을 추가하고 제거하는 방법을 알아 보았다. 그 뿐아니라, 새로운 보안 도메인을 적용하는 방법도 알아 보았다. tool 에 대한 나머지 부분은 부록 A 를 참고하기 바란다.

다음 장에서는 퍼포먼스를 향상 시키기 위해서, 보안 시스템을 어떻게 튜닝해야 하는지 알아 보겠다.

6 보안 시스템 튜닝

6.1 소개

본 장에서는 보안 시스템을 사용하더라도, JEUS 서버 전체에 미치는 퍼포먼스 영향력을 최소화 시키는 방법을 알아 보겠다.

여기서 설명하는 내용은 어느 정도 보안 시스템이 가져야 하는 특징과 충돌되는 부분도 있다. 기본적으로 퍼포먼스와 보안은 항상 타협의 대상으로, 보안이 강화되면 퍼포먼스가 떨어지는 경향이 있기 때문이다. 만약 보안이 퍼포먼스보다 더 중요한 고려 대상이라면 7 장을 자세히 읽어 보기 바란다.

6.2 보안 시스템의 퍼포먼스를 향상시키는 절차

6.2.1 소개

본 절에서는 보안 시스템의 전반적인 퍼포먼스를 향상시키는 방법을 살펴본다.

6.2.2 JEUS 클러스터에서 Secured Connection 을 사용하지 않는다.

Secured Connection 은 일반 Socket 에 암호화 된 packet 을 담아 사용하게 된다. 그러므로 일반 socket 통신을 할 때보다 더 많은 데이터를 전송하게 되어 Network 을 많이 사용하게 된다. JEUS 클러스터에서는 Secured Connection 을 사용하지 않도록 하는 것이 퍼포먼스를 향상시킨다.

6.2.3 J2SE SecurityManager 를 사용하지 않는다.

SecurityManager 를 사용하지 않으면 불필요한 연산을 수행하지 않게 된다. Security 를 사용하지 않는다는 말은 Authorization 을 사용하지 않는다는 말이다 (Authentication 은 반드시 사용해야 한다).

JEUSMain.xml 에서 <security-switch> 태그의 값을 off 로 설정을 하면 Authorization 기능을 사용하지 않게 된다. 따라서 Permission 을 체크할 때, 복잡하게 Permission 을 찾아서 체크하지 않고 언제나 true 를 리턴하게 되어 퍼포먼스가 향상된다. 하지만 이는 Security 를 사용하지 않겠다는 의미이기 때문에 Security 를 사용하지 않을 때에 사용하는 항목이다.

6.2.4 Non Blocking I/O 를 사용한다.

Non-Blocking I/O 를 사용하여 Network 의 block 현상을 줄인다. Non Blocking I/O 는 Network 을 통하여 Input/Output 이 발생할 때, 다음 Input/Output 이 발생하기를 기다리는 block 현상을 없앤것이다. 따라서 각 커넥션마다 I/O 를 위한 쓰레드가 별도로 필요하지 않기 때문에 CPU 사용량이나 FD 의 개수도 줄게된다. Non Blocking I/O 는 JEUS 에서 기본으로 사용하고 있다. 그러므로 Blocking I/O 보다 높은 퍼포먼스를 낸다.

6.3 결론

지금까지 보안 시스템의 전반적인 퍼포먼스를 향상 시키기 위해 취할 수 있는 몇 가지 방법을 알아 보았다.

다음 장에서는 보안 시스템에서 전반적인 보안 수준을 높이는 방법을 알아 보겠다.

7 보안 수준 향상

7.1 소개

본 장에서는 퍼포먼스를 희생하더라도 보안 시스템에 보안 수준을 최대로 끌어 올리는 방법에 대해 설명하겠다.

주의: 본 장은 속속들이 자세히 설명하지는 않고, 단순히 어떻게 하면 된다는 실마리를 제시해 줄 것이다.

7.2 보안 시스템의 보안 수준을 향상시키는 절차

7.2.1 소개

JEUS 의 전반적인 보안 수준을 끌어 올리기 위해 다음과 같은 방법들을 간단하게 검토해 보자.

- 전역 시스템 패스워드를 설정한다.
- JEUS 보안 클러스터에서 Secured Connection 을 사용한다.
- 저장된 Subject 와 Policy 를 권한이 없는 접근으로부터 보호한다.
- 보안 감사 메커니즘을 도입한다.
- J2SE SecurityManager 를 사용한다.
- third-party 보안 메커니즘을 사용한다.
- 운영 체제 자체에 보안을 설정한다.

주의: 이 세상에 100% 완벽을 장담하는 보안 솔루션은 없다는 것을 명심하기 바란다. 많은 시간과 비용을 들이면, 시스템을 망칠만한 구멍을 발견하는 것은 얼마든지 가능하다. 그러나, 보안 시스템의 보안을 강화할수록, 그런 구멍을 찾아내기는 더욱 힘들 것이고, 시스템을 망치기로 작정한 그 누군가는 그것이 과연 가치있는 일일지 고민하게 된다.

7.2.2 전역 시스템 변수 설정하기

3.7 에서 살펴 보았듯이 각 JVM 마다 시스템 전역 패스워들를 설정할 수 있다. 따로 설정하지 않으면 "globalpass"라는 디폴트 값이 적용되는데, 이를 이용해서 시스템에 접근하려는 불순한 의도를 가진 사람이 있을 수 있다.

따라서, 디폴트 값은 반드시 변경하고, 기밀을 철저히 유지해야 한다!

7.2.3 JEUS 보안 클러스터에서 Secured Connection 을 사용하기

네트워크 보안 서비스가 Secured Connection(보통 SSL/TLS 로 보호되는 connection)을 이용하도록 설정한다. 디폴트 네트워크 서비스를 사용할 때 어떻게 Secured Connection 을 설정하는지는 부록 F 를 참고하기 바란다.

이렇게 하는 목적은 누군가가 네트워크를 도청하고 있다가 민감한 정보를 채가는 것을 방지하고자 하는 것이다.

7.2.4 Subject 와 Policy 저장소 보호하기

Subject 와 Policy 저장소를 권한이 없는 접근으로부터 보호하는 것은 필수적인 일이다. 이는 어떤 종류의 저장소를 사용했느냐에 따라 방법이 달라 진다. 몇 가지 케이스를 살펴보자.

- 만약 데이터 베이스를 사용하고 있다면, Subject 와 Policy 를 저장하고 있는 테이블은 데이터베이스측에 설정된 인증과 권한부여 과정을 통해 보호될 수 있다. 또한 JEUS 보안 저장소와 데이터 베이스간의 커넥션을 SSL 같은 걸 사용해서 보호하도록 한다. 이를 설정하는 방법은 해당 데이터베이스 문서를 참조하도록 한다.

- 보안 관련 정보를 subjects.xml, policies.xml 과 같이 XML 파일에 저장해 두었다면, 이러한 파일에 접근할 수 있는 Permission 을 따로 만들어 두고, JEUS 보안 시스템과 administrators 만이 이 파일에 접근할 수 있도록 해야 한다. 만약 파일 저장소 가 어떤 종류든 간에 암호화를 지원한다면, 이를 사용한다. 어떻게 파일에 보안을 적용할 것인가에 대한 정보는 해당 저장소에 대한 문서를 참고하기 바란다. 그리고, 이 파일에 대한 읽기, 쓰기 권한을 설정하는 것은 해당 운영체제의 매뉴얼을 참고하기 바란다.

7.2.5 보안 감사 사용하기

시스템에서 발생한 보안 이벤트를 로그에 기록하고, 주기적으로 로그 내용을 살펴보면 전체적인 보안 수준을 향상시킬 수 있다. 다양한 보안 감사 메커니즘을 부록 F 에서 소개하고 있다.

그리고, 로그 파일을 권한이 없는 접근으로부터 보호해야 한다는 것을 명심하기 바란다.

주의: JEUS 보안 감사 메커니즘은 보통 `jeus.security.spi.EventHandlingService` SPI 클래스를 사용하여 구현된다.

7.2.6 J2SE SecurityManager 사용하기

JEUS 시스템의 견고성을 높이고, 악의적인 EJB 와 Servlet 코드로부터 JEUS 를 보호하기 위해서는 3.8 에서 설명한 대로 J2SE SecurityManager 를 설정하기 바란다.

7.2.7 Third-Party 보안 메커니즘 사용하기

민감한 정보가 다루어지는 실환경에서는 (은행 업무 어플리케이션 등) JEUS 본래의 보안 시스템과 더불어 방화벽이나 VPNs(Virtual Private Networks)와 같은 추가적인 보안 요소들을 설치하기를 강력하게 권장한다.

7.2.8 운영체제 보호하기

JEUS 가 동작하는 운영체제가 무엇이든 간에, 운영체제를 보호하는 몇가지 가이드라인이 있다.

- JEUS 클러스터로 묶인 머신들에 대한 물리적 접근을 제한한다. 예를 들면, 머신들을 한 곳에 몰아 두고, 자물쇠로 채운다.
- 신뢰가 확보된 administrator 만이 JEUS 파일과 프로세스에 접근할 수 있도록 프로세스 권한이나 파일 권한을 설정한다. 이는 해당 운영체제 매뉴얼을 참고하기 바란다.
- 최신 보안 패치가 적용된 운영 체제로 업그레이드 한다.
- 주기적으로 안티 바이러스 소프트웨어를 작동시킨다.
- 모든 보안 관련 문서들을 안전하게 보관한다. 예를 들면 패스워드가 적힌 문서를 자물쇠가 달린 캐비닛에 보관한다.

7.3 결론

지금까지 퍼포먼스가 떨어지는 비용을 감수하고서라도 JEUS 의 전체적인 보안을 향상시키는 몇 가지 방법에 대해 알아 보았다.

시스템의 보안과 퍼포먼스 사이에는 어느 정도 타협이 있기 마련이다.

다음 장에서는 JEUS 보안 시스템 API 를 사용하여 프로그래밍하는 방법을 알아 보겠다.

8 보안 시스템 API 프로그래밍

8.1 소개

본 장에서는 사용자 어플리케이션에 자신만의 특별한 보안 기능을 추가하기 위해, 보안 시스템 API 를 사용하여 프로그래밍하는 방법을 알아 보겠다.

이러한 예로는 어플리케이션을 통해 등록한 사용자를 자동으로 JEUS 보안 시스템에 등록하는 registration Servlet("auto-registration"이라고 한다)을 생각해 볼 수 있겠다.

주의: 어플리케이션 프로그래머는 보안 서비스를 개발하기 전에, 표준 J2EE 보안 모델과 JEUS 의 보안 서비스들이 원하는 보안 기능을 제공하는지 먼저 확인하기 바란다. 보안 API 를 사용하여 프로그램을 개발하게 되면 J2EE 서버간의 호환성이 떨어진다. 일반적으로 J2EE 서버간의 호환성을 유지하기 위해, 표준 J2EE 보안 인터페이스만 사용하기를 권장한다.

8.2 J2SE Permission 설정하기

악의적인 사용자 코드(가령, Servlet, EJB)로부터 JEUS 시스템을 보호하기 위해, 보안 API 를 사용할 수 있다.

사용자 코드 (Servlet, EJB 등..)에 보안 API 를 사용하는 경우는, 다음 세가지 경우 중 한가지 이상에 포함될 경우다.

- J2SE SecurityManager 를 사용하지 않는 경우이다(디폴트). 이 경우 LoginService.loginCodeSubject()가 사용되므로, 소스 코드는 보안 시스템의 모든 보안 체크를 그냥 통과하게 된다.
- J2SE SecurityManager 를 사용하지만, J2SE Policy 파일에, 소스 코드 (EJB, Servlet 등) 가 “java.security.SecurityPermission loginCodeSubject” 와 “java.security.SecurityPermission runTrustedLogin” J2SE Permission 을 가진다고 설정되어 있는 경우이다. 이런 경우에도 LoginService.loginCodeSubject()는 사용되므로, 소스 코드는 보안 시스템의 모든 보안 체크를 그냥 통과하게 된다.

- J2SE SecurityManager 를 사용하나, 소스코드 (Servlet, EJB)가 LoginService.login(Subject) 를 사용하여, 성공적으로 로그인한 경우이다. 이 때 Subject 는 타겟 보안 도메인의 subjects.xml 에 미리 설정되어 있는 것으로, policies.xml 에 설정된 필요한 JEUS Permission 을 가지고 있다.

J2SE SecurityManager 와 J2SE Policy 파일을 설정하는 방법은 3.8 절을 참고하기 바란다. subjects.xml 을 설정하는 방법은 3.5 절을 참고하기 바란다. JEUS 보안 시스템에 policies.xml 을 설정하는 방법은 3.6 과 부록 G 를 참고하기 바란다.

8.3 기본 API

어플리케이션 프로그래밍 레벨에서 보안 시스템과 연동할 때, jeus.security.base 패키지에 있는 다음 몇 개의 클래스가 매우 중요한 역할을 한다.

- jeus.security.base.Subject: 사용자를 나타낸다. Subject 는 단 하나의 메인 Principal 을 가지고 있으며, 메인 Principal 이 Subject 의 id(username)로 취급된다. jeus.security.base.Subject 클래스에 여러 개의 String 속성값이 전달되기도 한다.

- jeus.security.base.CredentialFactory: 앞서 소개한 Subject 클래스의 멤버 변수이며, Subject 에 대한 실제 Credential 을 생성하는데 사용된다. 가령, PasswordFactory 클래스는 패스워드 Credential 인스턴스를 생성하고, JKSCertificateFactory 클래스는 JKS keystore 로부터 인증서를 얻어 와서, 인증서 Credential 인스턴스를 생성한다.

- jeus.security.base.Policy: 하나의 principal-to-role 매핑과, 여러 개의 role-to-resource 매핑을 나타낸다. PermissionMaps 을 멤버 변수로 가지고 있다.

- jeus.security.base.PermissionMap: java.security.Permission 인스턴스들의 컨테이너로 볼 수 있으며, Policy 클래스의 멤버 변수이다.

- jeus.security.base.Role: 논리적 Role 을 나타내는 인터페이스이다. role-to-resource PermissionMap 에서는 Role 인스턴스가 Resource Permission 에 매핑된다. 마찬가지로 principal-to-role PermissionMap 에서는 Principal 이 Role Permission 에 매핑된다.

- jeus.security.base.SecurityException: 보안 위반시 - 로그인 실패, 인증 실패, 권한 체크 실패등.- 발생하는 예외
- jeus.security.base.ServiceException: 보안 시스템에서 심각한 런타임 에러가 발생할 때 발생하는 예외.

해당 클래스에 대해 더 자세한 정보는 Javadoc 과 부록 J 를 참고하기 바란다.

8.4 리소스 API

리소스와 관련해서, jeus.security.base 패키지에 있는 몇 몇 기본 클래스뿐 아니라, jeus.security.resource 패키지에 있는 다음 클래스들이 중요하다.

- jeus.security.resource.PrincipalImpl: java.security.Principal 인터페이스의 구현 클래스
- jeus.security.resource.GroupPrincipalImpl: 그룹 Principal 을 나타내는 PrincipalImpl 의 서브 클래스
- jeus.security.resource.Password: 단순한 Password Credential 인스턴스로, PasswordFactory 에의해 생성된다.
- jeus.security.resource.PasswordFactory: Password Credential 을 생성하는 CredentialFactory.
- jeus.security.resource.Lock: 일종의 Credential 로, 해당 Subject 에 락을 건다. 락이 걸린 Subject 로 로그인하게 되면, 항상 실패하게 된다.
- jeus.security.resource.LockFactory: Lock Credential 을 생성하는 CredentialFactory.
- jeus.security.resource.ExpiryTime: 일종의 Credential 로 해당 Subject 의 만료기간을 설정한다. Subject 가 만료된 후, 해당 Subjet 를 사용하여 로그인을 시도하면, 모두 실패한다.
- jeus.security.resource.ExpiryTimeFactory: ExpiryTime 을 생성하는 CredentialFactory
- jeus.security.resource.RoleImpl: Role 인터페이스를 구현한 클래스

- jeus.security.resource.RolePermission: 특정 Principal 이 특정 Role 에 속한다는 것을 나타내는 java.security.Permission 의 서브 클래스. RolePermission 은 principal-to-role 매핑을 나타내는데 사용된다.

- jeus.security.resource.TimeConstrainedRolePermission: RolePermission 의 서브 클래스로, 현재 시간이 설정된 시간 범위내에 있을 때만 Principal 이 이 클래스의 수퍼클래스가 나타내는 Role 에 속한다는 것을 나타낸다.

- jeus.security.resource.ResourcePermission: java.security.Permission 의 서브 클래스로, Role 이 리소스에 접근해서 특정 액션을 실행할 수 있다는 개념을 나타낸다. 따라서 ResourcePermission 은 role-to-resource 매핑을 나타내는데 사용된다.

해당 클래스에 대한 자세한 정보는 Javadoc 과 부록 J 를 참고하기 바란다.

8.5 SPI

보안 시스템의 근간을 이루는 서비스와 작업하려면, jeus.security.spi 패키지에 있는 SPI 클래스를 사용해야 한다.

- jeus.security.spi.LoginService: 해당 Subject 를 인증하고, Subject 에 대한 Permission 을 체크한다.

- jeus.security.spi.AuthenticationRepositoryService: Subject 저장소로부터 Subject 를 추가, 삭제, 조회하는데 사용된다. 이 API 를 이용하면 Subject(user)를 프로그램 내에서 추가할 수 있다.

- jeus.security.spi.AuthorizationRepositoryService: Policy 저장소로부터 Policy 데이터를 추가, 삭제, 조회하는데 사용된다. 이 API 를 이용하면 프로그램내에서 Permission 을 추가할 수 있다.

해당 클래스에 대한 자세한 정보는 Javadoc 과 부록 J 를 참고하기 바란다. SPI 클래스에 대한 더욱 상세한 정보는 9 장을 참고하기 바란다.

8.6 보안 프로그램 예제

다음 코드는 보안 API 를 사용한 프로그램의 일부이다.

```
// Login the CodeSubject so that security checks are
// disabled (so that we can modify the Subject and Policy
// stores)
LoginService.loginCodeSubject();

// Make Subject with Principal “pete”
Principal petePrincipal = new new PrincipalImpl(“pete”);
Subject pete = new Subject(petePrincipal);

// Make password “petepw” for Subject “pete”
PasswordFactory pf = new PasswordFactory(“petepw”);
pete.getCredentialFactories().add(pf);

// Add new Subject to the Subject store
AuthenticationRepositoryService.addSubject(pete);

// Make a new Policy
Policy policy = new Policy();

// Make role “someRole”
Role someRole = new RoleImpl(“someRole”);

// Make a RolePermission for role “someRole”
Permission rolePermission = new RolePermission(someRole);

// Add the RolePermission for “someRole” to the Policy
policy.getRolePolicy().addPermission(
     rolePermission, new Object[] {petePrincipal}, false, false);

// Create a ResourcePermission for resource “rsc1” with actions
// “action1” and “action2”
Permission rscPermission =
      new ResourcePermission(“rsc1”, “action1,action2”);

// Add the ResourcePermission to the Policy using
// context id “ctx1”
policy.getResourcePolicy(“ctx1”, true).addPermission(
     rscPermission, new Object[] {someRole}, false, false);
```

```
// Add the new Policy to the Policy store
AuthorizationRepositoryService.addPolicy(policy);

// Logout the CodeSubject so that security checks are
// enabled again
LoginService.logout();

// Make a Subject to be logged in
Subject pete2 = Subject.makeSubject(“pete”, “petepw”);

// Login Subject “pete” (should succeed since we added
// “pete” earlier)
LoginService.login(pete2);

// Check ResourcePermission “rsc1” for current Subject (“pete”)
// Should succeed since we added Policy for this above
LoginService.checkPermission(
     “ctx1”, new ResourcePermissin(“rsc1”, “action2”);

// Print the name of the current Subject (“pete”)
System.out.println(
     LoginService.getCurrentSubject().getPrincipal().getName());

// Logout “pete”
LoginService.logout();
```

8.7 결론

지금까지 JEUS 보안 시스템에서 보안 API 를 사용하여 프로그래밍하는 방법에 대해 알아 보았다. 구체적으로 기본 보안 API, 리소스 API, SPI API 에 대해 알아 보았다. 마지막으로 JEUS 보안 시스템에서 실제 적용할 수 있는 프로그램 예제를 작성해 보았다. 해당 주제에 관한 더 자세한 정보는 Javadoc 과 부록 J 를 참고하기 바란다.

다음 장에서는 다양한 SPI 클래스를 구현하여, JEUS 보안 시스템에 새로운 보안 서비스를 추가하는 방법에 대해 알아 보겠다.

9 커스텀 보안 서비스 개발하기

9.1 소개

본 장에서는 JEUS 보안 시스템의 주요 특징인, 커스텀 보안 서비스를 개발하는 방법에 대해 살펴 보겠다.

이를 이용하면, JEUS 보안 시스템과 다양한 외부 보안 시스템, 보안 데이터 저장소를 쉽게 통합할 수 있다.

본 장에서는 커스텀 보안 서비스를 개발하는 것과 관련된 몇가지 중요한 개념에 대해 살펴 보겠다.

- Sevice 클래스
- 구현의 기본 패턴
- 12 개의 SPI 클래스의 개요
- 12 개의 SPI 클래스에 대한 자세한 설명
- 12 개의 SPI 클래스간의 의존도
- 커스텀 보안 서비스 설정 방법

9.2 Service 클래스

9.2.1 소개

pluggable 보안 아키텍쳐에서 가장 기본이 되는 클래스는 jeus.security.base.Service 클래스이다.

Service 클래스는 보안 서비스를 구현하려는 클래스들이 반드시 확장해야 하는 추상 클래스로, 현재 jeus.security.spi 패키지에 있는 모든 SPI 클래스도 이 클래스를 확장한 것이다.

Service 클래스는 모든 보안 서비스에 공통적으로 적용되는 항목들을 포함하고 있다.

- Description
- Domain
- Type
- Name
- MBean
- State
- Properties

지금부터 각 항목에 대해 자세히 살펴보겠다.

9.2.2 Service 클래스 다이어그램

[그림 16]는 Service 클래스의 클래스 다이어그램이다.

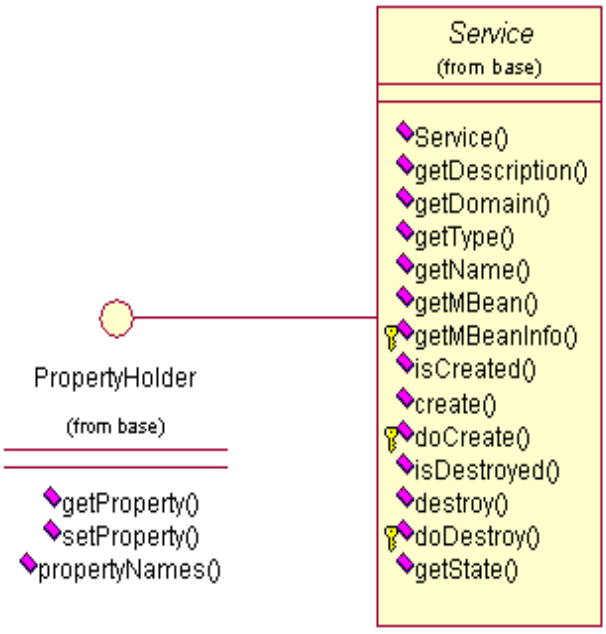

그림 16 Service 클래스 다이어그램

9.2.3 Service Description

Service 클래스는 제공하려는 서비스가 무엇인지에 관해 간단하게 설명하는, String 타입의 description 을 제공한다.

Service 에 대한 설명은 getDescription() 메소드로 가져올 수 있다.

9.2.4 Service Domain

런타임에 생성된 인스턴스는 특정 도메인에 할당된다. 따라서 Domain 은 본질적으로 Service 인스턴스의 집합이라 할 수 있다.

Service 의 도메인은 getDomain() 메소드로 가져올 수 있다.

9.2.5 Service Type

Service 구현 클래스는 단 하나의 type 을 가진다. type 이란 Service set 으로부터 특정 서비스 인스턴스를 가져 오기 위해 사용하는 일종의 마크이다.

가령, 특정 Service 구현 클래스가 인증 기능을 제공한다고 하자. 다른 보안 서비스에서 이 인증 Service 클래스를 사용하려면, 도메인에 해당 클래스를 요청해야 한다. 이 때 Service type 을 넘겨 주어서, 도메인이 정확한 Service 인스턴스를 찾아서 리턴하도록 만들 수 있다.

Service 타입은 getType() 메소드로 가져올 수 있다. 그러나, getType() 메소드는 추상 메소드이므로 반드시 서브 클래스에서 구현해야 한다.

Service type 은 실제로는 java.lang.Class 타입으로 jeus.security.spi 패키지에 포함되어 있는 SPI 클래스의 Class 인스턴스이다. 각 SPI 클래스에서 getType() 메소드를 final 로 구현하고 있다(즉, SPI 의 서브 클래스에서 는 getType() 메소드를 재정의할 수 없다).

9.2.6 Service Name

Service 의 명칭은 getName() 메소드로 가져올 수 있다.

9.2.7 Service MBean

Service 클래스는 특정 Service 인스턴스를 관리하는데 사용되는, JMX MBean 과 결합되어 있다.

MBean 은 getMBean() 메소드로 가져올 수 있으며, 서브 클래스에서 디폴트 MBean 이 아닌 다른 MBean 을 리턴하려면, 이 메소드를 재정의해야 한다.

기본적으로 MBean 은 protected 메소드로 선언된 getMBeanInfo()로부터 리턴된 MBeanInfo 를 바탕으로 생성된다. 서브 클래스는 이 메소드를 재정의해서 디폴트 이외의 다른 MBeanInfo 인스턴스를 리턴하도록 할 수 있다. 메소드 재정의시, 수퍼 클래스의 MBeanInfo 도 포함하도록 신경써야 한다.

9.2.8 Service State

Service 구현 클래스는 created 또는 destoryed 둘 중 하나의 상태를 가진다 상태는 create() 와 destroy() 메소드를 호출하여 변경할 수 있다. 이 메소드들은 추상 메소드인 doCreate()와 doDestroy() 메소드를 호출하여 실제 업무를 위임한다. 따라서, Service 구현 클래스는 doCreate()와 doDestroy() 메소드를 반드시 구현해야 한다.

doCreate()와 doDestroy() 메소드 내에 서비스를 적절히 초기화 하고, 소멸하는 코드가 포함되어 있다. 전형적인 doCreate() 메소드는 DB 커넥션과 같은 자원을 획득하는 코드가 포함되어 있고, doDestroy() 메소드에는 이러한 획득된 자원을 해제하는 코드가 포함되어 있다.

Service 클래스의 현재 상태는 isCreate() 와 isDestroyed() 메소드를 호출하여 알 수 있다. 또한 현재 상태를 String 으로 리턴하는 getState() 메소드를 가지고 있다.

[그림 17]에서 Service 클래스의 상태 차트를 보여주고 있다.

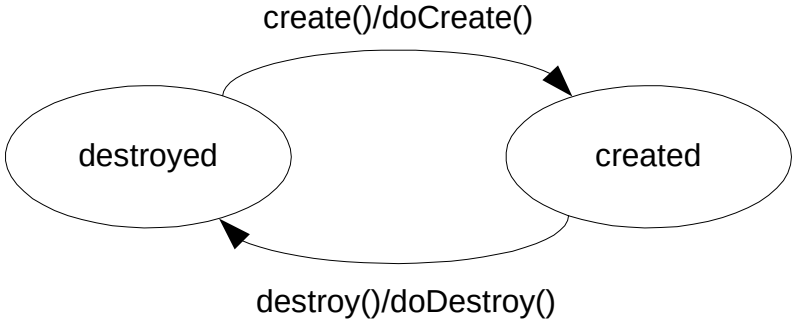

그림 17 Servlet 의 두가지 상태

9.2.9 Service Properties

Service 클래스는 name-value 쌍의 속성을 가지고 있다. 이러한 속성을 설정하고, 가져 오는 작업은 jeus.security.base.PropertyHolder 인터페이스를 통해 이루어 진다. Service 구현 클래스는 이 인터페이스를 구현해야 한다.

속성들은 Service 인스턴스를 초기화하는데 사용된다. Service 인스턴스가 생성되고 create() 메소드가 불려지기 전에, 속성들이 설정된다. 이후, doCreate()

메소드가 불려지면, 설정된 속성 값을 쿼리해 와서 Service 인스턴스를 적절하게 초기화 한다.

9.2.10 결론

지금까지 SPI 클래스가 반드시 상속해야 하는 Service 클래스에 대한 기본 사항을 알아 보았다.

다음 절에서는, 중요한 SPI 클래스에 대해 간략하게 살펴 보겠다.

9.3 커스텀 보안 서비스를 구현하는 기본적 패턴

커스텀 보안 서비스를 구현하기 위해, 다음과 같은 절차를 따르는 것이 일반적이다.

1. 사전에, 보안 시스템과 보안 시스템 아키텍처를 충분히 이해하고 있어야 한다.
2. 커스텀 보안 서비스가 어떤 보안 기능을 제공해야 하는지 파악한다.
3. 해당 특성을 가지고 있는 SPI 클래스를 선택한다. 다음 절에서 주요 SPI 클래스에 대한 자세한 정보를 알아 보겠다.
4. 해당 SPI 클래스에 대한 문서를 주의깊게 살펴본다 (Javadoc, 부록 **J**, 다음 절에 나오는 주요 SPI 클래스에 대한 설명을 참고한다)
5. 해당 SPI 클래스의 서브 클래스를 작성하고, 서브 클래스에서 다음 메소드들을 구현한다. 그리고, 반드시 파라미터가 없는 public 생성자를 제공해야 한다.
    a. 선택적으로 Service 초기화에 사용되는 속성들을 정의한다. 각 속성은 public static final String 타입으로, 각 각이 무엇을 나타내는 지는 문서화 되어야 한다.
    b. doCreate() 메소드를 구현한다. 이 메소드는 보안 서비스가 시작할 때 단 한번 불려 지며, 자원 할당과 같은 일반적인 초기화 작업을 수행 한다. 이 메소드는 getProperty() 메소드를 통해 설정 값을 읽어 들인다. getProperty() 메소드의 파라미터는 이전 단계 a 에서 설정된 속성명이다.

c. doDestroy() 메소드를 구현한다. 이 메소드는 서비스가 종료하기 전에 단 한번 호출된다. 이 메소드는 doCreate() 메소드에서 할당된 자원을 해제하거나, 특정 파일에 로그를 남기는 등의 일반적인 clean-up 작업을 수행한다.

d. 선택한 SPI 클래스의 모든 추상 메소드를 Javadoc 에서 설명하고 있는 방식대로 구현한다.

e. 선택적으로 JMX 를 통해 서비스를 관리하는데 사용되는 메소드들을 구현할 수 있다. getMBean() 또는 getMBeanInfo() 메소드를 구현한다.

6. SPI 를 구현한 클래스를 컴파일한다.

7. 3 장에서 설명한 대로, 새로운 보안 서비스를 JEUS 에 등록한다.

다음 절에서 jeus.security.spi 패키지에 포함되어 있는 다양한 SPI 클래스에 대해 설명하겠다.

9.4 SPI 클래스

다음 SPI 클래스들이 jeus.security.spi 패키지내에 정의되어 있다.

Table2. jeus.security.spi 패키지에 있는 표준 SPI 클래스

SPI class name:	Purpose	Cardinality*
`SecurityInstaller`	보안 시스템을 설치하고 언인스톨한다.	1
`LoginService`	Subject 로 로그인하고 로그아웃 한다.	1
`SubjectValidationService`	로그인 전에 해당 Subject 의 Credential 이 유효한지 아닌지 체크한다.	0 개 이상

SPI class name:	Purpose	Cardinality*
SubjectFactoryService	로그인 전에 커스텀 Subject 를 생성한다.	1
AuthenticationService	로그인 전에 Subject 를 인증한다.	0 개 이상
AuthenticationRepositoryService	Subject 를 Subject 저장소로부터 추가,삭제,조회 한다.	1
CredentialMappingService	Credential 을 Subject 에 매핑한다.	0 개 이상
CredentialVerificationService	Subject 가 가지고 있는 Credential 중 최소한 하나라도 유효한지 검증한다.	0 개 이상
AuthorizationService	Subject 가 특정 Role 또는 Resource 에 접근할 권한이 있는지 체크한다.	1 개 이상
AuthorizationRepositoryService	Policy 를 Policy 저장소로부터 추가,삭제,조회한다.	1
EventHandlingService	보안 이벤트 에 대한 커스텀 이벤트 핸들러, 다양한 보안 감사를 구현한다.	0 개 이상

* "cardinality"는 각 도메인에 해당 SPI 클래스가 몇 개나 있을 수 있는지 알려준다 (예외로 SecurityInstaller 는 전체 JVM 에 대해 오로지 하나만 있어야 한다).

다음 절에서 각 SPI 클래스에 대해 자세히 설명하겠다. SPI 클래스에 대한 더욱 상세한 설명은 부록 J 를 참고하기 바란다.

9.5 SecurityInstaller SPI

jeus.security.spi.SecurityInstaller SPI 는 보안 시스템을 인스톨하는 작업을 커스터마이징 하는데 사용된다. 가령, 디폴트 SecurityInstaller 는 XML 파일에서 설정 정보를 읽어오지만, 데이터베이스를 사용하거나, 프로그램 내에서 직접 설정 정보를 제공하는 SecurityInstaller 도 구현할 수 있을 것이다.

jeus.security.spi.SecurityInstaller.installSecurity(Environment) 메소드는 JVM 이 시작할 때, 단 한번 호출된다. 이 메소드는 다시 추상 메소드인 doInstallSecurity(Environment)를 호출한다. 구현 클래스는 이 메소드 내에서 보안 시스템을 인스톨하는 로직을 작성한다. 즉 메소드는 jeus.security.base.Domain 인스턴스들을 생성하고, 각 Domain 에 jeus.security.base.Service 인스턴스를 추가하는 로직을 포함하고 있어야 한다.

installSecurity() 메소드의 파라미터인 jeus.security.base.Environment 는 보안 시스템이 작동되는 환경에 관한 정보를 포함하고 있다 (가령, 로컬 JVM 이 client 인지 server 인지 알려주는 정보를 포함하고 있다)

jeus.security.spi.SecurityInstaller.uninstallSecurity() 메소드는 JVM 이 종료되기 직전에 단 한번 호출된다. 이 메소드는 doUninstallSecurity() 추상 메소드를 호출한다. 구현 클래스는 이 메소드내에서 보안 시스템을 언인스톨하는 로직을 작성한다. 즉 메소드는 jeus.security.base.Domain 인스턴스를 없애고, 디스크에 설정 정보를 백업 하는등의 작업을 포함한다.

SecurityInstaller 는 일반적인 보안 서비스와는 다른 방법으로 설치된다. SecurityInstaller 를 설정하는 방법은 3.4 절에 설명되어 있다.

주의: 보안 시스템이 어떻게 작동하는지에 대한 충분한 지식이 없는 상태에서, 커스텀 SecurityInstaller 로 디폴트 SecurityInstaller 를 교체하는 것은 바람직하지 않다. 대부분의 경우, 디폴트 SecurityInstaller 만으로도 충분하다. 만약 SecurityInstaller 를 반드시 교체해야 한다면, 최소한 jeus.security.impl.installer.DefaultSecurityInstaller 를 상속하고, 반드시 재정의가 필요한 메소드만 재정의한다.

JVM 당 단 하나의 SecurityInstaller 인스턴스가 설정되어 있어야 한다.

해당 SPI 클래스에 대한 자세한 정보는 Javadoc 과 부록 ”J”를 참고하기 바란다.

9.6 LoginService SPI

jeus.security.spi.LoginService SPI 클래스의 목적은 Subject 를 현재 실행중인 쓰레드에 결합시키는 메커니즘을 제공하는 것이다.

일반적인 LoginService 메커니즘은 Subject 를 쓰레드에 결합 시키기 전에 먼저 Subject 가 성공적으로 인증된 것인지 체크한다. 이는 보통 jeus.security.spi.AuthenticationService.authenticate(Subject subject)메소드를 통해 이루어진다.

이전 장에서 언급한 대로, LoginService 는 stack-based 로그인 메커니즘을 사용하고 있다. 이는 2.5.1 절에서 자세히 설명했다.

뿐만 아니라, LoginService 는 현재 로그인된 Subject 의 권한을 체크하는 메소드도 제공한다. 이 메소드는 jeus.security.spi.AuthorizationService.authorize(Permission p, String contextId, Subject subject)로, Checked Permission 과 contextId, 현재 로그인된 Subject 를 파라미터로 넘겨 받는다.

도메인당 반드시 하나의 LoginService 가 설정되어 있어야 한다.

해당 SPI 에 대한 자세한 정보는 Javadoc 과 부록 ”J”를 참고하기 바란다.

9.7 SubjectValidationService SPI

jeus.security.spi.SubjectValidationService SPI 는 Subject 가 가지고 있는 Credential 이 유효한지 아닌지를 체크하는데 사용된다. 이 말은 때때로 Credential 이 유효하지 않는 경우도 있음을 나타낸다(유효하지 않는 Credential 은 곧 Subject 가 유효하지 않다는 말이고, 결과적으로 해당 Subject 로는 로그인할 수 없다는 것을 나타낸다).

SubjectValidationService 의 전형적인 예는, Subject 가 "lock" Credential 을 가지고 있는지 없는지 체크하는 것이다. 만약 "lock" Credential 을 가지고 있다면, Subject 는 락에 걸린 것으로 취급되어 더 이상의 로그인 프로세스가 진행되지 않는다. SubjectValidationService.checkValidity(Subject)메소드는 보통 LoginService 구현 클래스에서 호출된다. 만약 이 메소드에서 SecurityException 이 발생되면, 모든 로그인 과정은 실패로 돌아간다.

(Subject 에 "lock" Credential 을 자동으로 설정하기 위해서, EventHandlingService 가 사용된다. 이에 관한 자세한 설명은 9.15 절을 참고하기 바란다).

인증(파라미터로 넘어온 Subjec 가 실제 해당 Subject 인지 검증하는 것)과 Subject 유효성 검사는 서로 별개의 것이다. 그러나, 로그인 과정에는 두 가지 서비스가 모두 사용된다.

주의: SubjectValidationService SPI 는 CredentialVerificationService 와 많은 점에서 비슷해 보이지만, 기능명에서 다르다: SubjectValidationService 는 Credential 을 체크해서, Subject 가 유효한지 아닌지를 판단하고, CredentialVerificationService 는 해당 Subject 가 실제 Subject 인지 증명하는 Credential 을 최소한 하나라도 가지고 있는지를 체크한다. 따라서, SubjectValidationService 는 LoginService 에서 Subject 가 성공적으로 인증된 후 사용되고, 반면에 CredentialVerificationService 는 AuthenticationService 에서 인증 과정 중에 사용된다.

도메인당 0 개 이상의 SubjectValidationService 인스턴스가 설정될 수 있다. 전체 SubjectValidationService 에서 SecurityException 이 하나도 발생하지 않으면, Subject 는 유효하다고 판단되고, 로그인 과정이 계속 진행된다. 그러나, 만약 SecurityException 이 한 군데서라도 발생하면, 전체 유효성 검사는 실패로 돌아 가고, 로그인 과정은 더 이상 진행되지 않는다.

해당 SPI 클래스에 대한 자세한 정보는 Javadoc 과 부록 ”J”를 참고하기 바란다.

9.8 SubjectFactoryService SPI

jeus.security.spi.SubjectFactoryService SPI 클래스는 외부에서 제공하는 정보없이 Subject 를 생성하는 특별한 SPI 이다.

SubjectFactoryService SPI 는 보통 LoginService.doLogin() 메소드 내에서 호출된다. doLogin() 메소드는 어떤 Subject 나 Credential 도 파라미터로 넘겨 받지 않고, 대신에 로그인된 Subject 를 SubjectFactoryService 서비스를 통하여 생성한다.

일반적으로, LoginService.doLogin() 메소드는 client 코드에서만 사용된다(가령, application client container)

전형적인 구현 클래스는 프롬프트로부터 사용자명과 패스워드를 입력받아, 이 정보를 기초로 Subject 를 생성해서 리턴한다.

SubjectFactoryService 는 로그인 메커니즘에서 패스워드 이외의 Credential 타입도 지원하기 위해 만들어 졌다.

사용자 코드에서 LoginService.login() 메소드를 호출할 때, 단 하나의 SubjectFactoryService 인스턴스만이 설정되어 있어야 한다.

해당 SPI 에 대한 자세한 정보는 Javadoc 과 부록 "J" 를 참고하기 바란다.

9.9 AuthenticationService SPI

jeus.security.spi.AuthenticationService 는 Subject 를 인증하기 위한 클래스이다. 즉, 파라미터로 넘겨받은 Subject 가 Subject 저장소에 등록되어 있는 실제 Subject 와 일치하는지 검증하는 것이다.

인증 과정은 다음과 같다. 우선 jeus.security.spi.CredentialMappingService.getSubjectName(Object)를 호출하여 Subject 의 Credential 이 어떤 사용자에 맵핑되는지 확인한다. 그리고 나서 jeus.security.spi.AuthenticationRepository.getSubject(String username) 메소드를 호출하여 Subject 저장소로부터 실제 등록되어 있는 Subject 를 로컬로 복사해온다. (이를 로컬 Subject 라 하자). 만약 로컬 Subject 가 null 이 아니라면, 로컬 Subject 의 Credential 과 인증할 Subject 의 Credential 이 일치하는지 비교해 본다. 이 비교는 jeus.security.spi.CredentialVerificationService.verifyCredentials(Subject, Subject)메소드를 호출하여 수행된다(몇 몇 경우에는 equals(Objet)를 사용하여 Credential 을 비교하기도 하는데, 이는 유연한 방법이 아니다).

마지막으로, 위의 과정이 모두 성공적으로 완료되면, 로컬 Subject 가 authenticate(Subject)메소드로부터 리턴되고, 결국 이 Subject 를 사용하여 LoginService 내에서 로그인한다(이전 절을 참고하기 바란다).

도메인당 한 개이상의 AuthenticationService 를 설정할 수 있다. 만약 설정된 AuthenticationService 중 최소한 하나라도 Subject 를 성공적으로 인증했다면, 모든 AuthenticationService 가 인증한 것으로 간주한다. 즉 단 하나의 AuthenticationService 라도 성공적으로 Subject 를 인증했다면, 추가적으로 다른 AuthenticationService 에 쿼리를 던지지 않는다.

해당 SPI 클래스에 대한 자세한 설명은 Javadoc 과 부록 "J"를 참고하기 바란다.

9.10 AuthenticationRepositoryService SPI

jeus.security.spi.AuthenticationRepositoryService SPI 는 Subject 저장소로부터 Subject 를 추가, 삭제, 조회하는 메소드를 가지고 있다.

이 SPI 는 AuthenticationService 를 구현하는 클래스에서 사용되거나, J2EE 어플리케이션 코드내에서 직접 사용될 수 있다. 가령 웹사이트를 통해 새로운 사용자가 등록할 때 마다, 자동으로 Subject 를 Subject 저장소에 추가하는 코드를 작성하려면, 이 SPI 를 사용해야 한다.

원래 AuthenticationRepositoryService 의 가장 큰 목적은 런타임에 jeus.security.base.Subject 인스턴스를 특정 저장소 타입에 저장하는 것이다. 전형적인 저장소로는 XML 파일이나 데이터 베이스가 있다.

일반적으로 AuthenticationRepositoryService SPI 를 별도로 구현할 필요는 없다. 그러나, 디폴트 AuthenticationService 를 사용하거나 (사용자 인증과정에서 AuthenticationRepositoryService.getSubject(String) 메소드를 호출하기 때문이다), J2EE 어플리케이션 코드에서 직접 사용할 때는 반드시 구현해야 한다.

도메인당 단 하나의 AuthenticationRepositoryService 인스턴스만 설정될 수 있다(현재 1 개 이상의 AuthenticationRepositoryService 를 설정하는 것은 불가능하다).

해당 SPI 에 대한 자세한 설명은 Javadoc 과 ”J”절을 참고하기 바란다.

9.11 CredentialMappingService SPI

jeus.security.spi.CredentialMappingService 는 Subject 의 사용자명(username)이 제공되지 않는 경우에 필요하다(즉, 메인 Principal 이 null 일 때이다). 이 경우에는 Subject 로부터 Credential 을 가져와서, Credential 이 어느 사용자명에 매칭되는지 알아 보게 된다. 매칭된 사용자명은 이후 인증 과정에 사용된다.

전형적인 예로 java.security.Certificate 인스턴스를 들 수 있다. 이 인스턴스는 Certificate Credential 을 사용하므로, 메인 Principal 을 가지고 있지 않다. 이 경우 인증 과정동안, Sujbect 로부터 Certificate Credential 을 얻어 와서, CredentialMappingService.getSubjectName(Object) 메소드에 파라미터로 넘긴다. 해당 메소드는 다시 CredentialMappingService.doGetSubjectName(Object)를 호출한다. 이 메소드 내부에서 Certificate 저장소로부터 Certificate 를 통해 사용자명을 역추적하고, 결국 매칭되는 사용자 명을 리턴한다.

9.12 CredentialVerificationService SPI

jeus.security.spi.CredentialVerificationService SPI 는 새로운 타입의 Proof Credential 을 지원하기 위해 사용된다. proof Credential 이란 파라미터로 넘어온 Subject 가 실제 존재하는 Subject 인지 검증하기 위해 사용되는 Credential 을 말한다. Proof Credential 의 전형적인 예가 바로 패스워드이다 (jeus.security.resource.Password 클래스로 구현된다).

CredentialVerificationService SPI 는 서브 클래스가 반드시 구현해야 하는 단 하나의 메소드인 doVerifyCredentials(Subject, Subject)를 선언하고 있다. 이 메소드 시그너쳐에서 첫번째로 나오는 Subject 는 "reference Subject"라 하고, Subject 저장소에 등록되어 있는 실제 Subject 의 Credential 들을 가지고 있다. 두 번째로 나오는 Subject 는 "proof Subject"라 하고, "proof Credential"들을 가지고 있다. doVerifyCredentials(Subject, Subject)메소드는 두개의 Subject 의 Credential 들을 서로 비교해서 하나라도 일치하는 것이 있는지 확인한다. Credential 들간의 매칭은 다양한 방법으로 이루어 진다(보통 equals(Object)메소드만으로는 충분하지 않다). 만약 두 Subject 간에 하나라도 일치하는 Credential 이 있다면, 메소드는 리턴되고, 그렇지 않다면, jeus.security.base.SecurityException 이 발생한다.

주의: 몇 몇 경우에는, reference Subject 에 대한 정보가 필요없을 때가 있다. 즉 특정 Proof Credential 의 경우 Proof Subjet 에 대한 정보만 가지고 Credential 을 검증할 수 있다(가령 특정 Certificate Credential 의 경우가 그렇다).

jeus.security.resource.Password 을 매치하는 CredentialVerificationService 는 jeus.security.impl.verification.PasswordVerificationService 클래스이다

도메인당 0 개 이상의 CredentialVerificationService 를 설정할 수 있다. 만약 최소한 하나의 CredentialVerificationService 에서라도 매칭이 확인되면, 전체 CredentialVerificationService 가 모두 성공한 것으로 간주된다. 그렇지 않으면, SecurityException 이 던져지고, 전체 검증은 모두 실패한 것으로 간주된다. (결과적으로 인증 과정이 실패하게 된다).

해당 SPI 에 대한 자세한 정보는 Javadoc 과 부록 ”J”를 참고하기 바란다.

9.13 AuthorizationService SPI

The jeus.security.spi.AuthorizationService SPI 는 보안 시스템에서 권한 체크와 관련된 메소드를 정의하고 있다. 이 SPI 의 목적은 다음과 같은 질문에 답하는 것이다 " Subject S 는 A 라는 액션을 실행할 권한이 있는가?"

전형적인 구현 클래스는 jeus.security.spi.AuthorizationRepositoryService.getPolicy(contextId) 메소드를 호출하여, 그 결과로 리턴된 jeus.security.base.Policy 를 분석해서 위의 질문에 대한 답을 구한다. AuthorizationRepositoryService SPI 를 사용하지 않는 다른 종류의 구현도 얼마든지 가능하다.

도메인당 1 개 이상의 AuthorizationService 가 설정될 수 있다. 최소한 하나의 AuthorizationService 에서라도 양수값을 리턴하면, 모든 권한 체크 과정을 통과한 것으로 간주한다. 즉 하나의 AuthorizationService 에서라도 Permission 이 허가 되었다면, 추가적으로 다른 AuthorizationService 에 권한을 확인하는 쿼리를 던지지 않는다.

해당 SPI 에 대한 자세한 정보는 Javadoc 과 부록 "J"를 참고하기 바란다.

9.14 AuthorizationRepositoryService SPI

jeus.security.spi.AuthorizationRepositoryService SPI 는 Policy 저장소로부터 jeus.security.base.Policy 인스턴스를 추가, 제거, 조회하는 메소드를 포함하고 있다.

이 SPI 는 AuthorizationService 구현 클래스에서 사용되거나, J2EE 어플리케이션 코드 내에서 특정 이벤트가 발생할 때 직접 Policy 를 Policy 저장소에 추가하거나 제거하기 위해 사용한다.

원래 AuthorizationService 의 가장 큰 목적은, 런타임에 jeus.security.base.Policy 를 특정 타입의 Policy 저장소에 저장하는 것이다. 전형적인 구현 클래스는 XML 파일이나, 데이터베이스, LDAP 서버에 Policy 를 저장한다.

AuthorizationRepositoryService SPI 를 구현해서 사용하는 것은 선택사항이다. 그러나, 디폴트 AuthorizationService 서비스를 사용한다면, 이 서비스가 AuthorizationRepositoryService.getPolicy(String) 메소드를 호출하기 때문에, 반드시 구현해야 한다.

도메인당 단 하나의 AuthorizationRepositoryService 만 설정될 수 있다(현재 한 개이상의 AuthorizationRepositoryService 를 도메인에 설정하는 것은 불가능하다).

해당 SPI 에 대한 자세한 설명은 Javadoc 과 부록 "J"를 참고하기 바란다.

9.15 EventHandlingService SPI

EventHandlingService SPI 는 각 종 보안 이벤트를 캡쳐해서 처리하는 인터페이스이다.

EventHandlingService SPI 를 사용하면, 보안 이벤트가 발생할 때, 로그를 남기고, 관리자에 자동으로 메일을 보내고, Subject 에 락을 거는 등의 작업을 쉽게 구현할 수 있다.

기본 개념은 매우 간단한데, 보안 서비스에서 발생한 jeus.security.base.Event 타입의 이벤트는 같은 보안 도메인에 설정되어 있는 모든 EventHandlingService 들에 통보된다. 그러면, EventHandlingService 는 이벤트의 내용에 따라 각각 다르게 처리한다.

EventHandlingService 의 예로 Subject 에 락을 거는 경우를 생각해 볼 수 있다. AuthenticationService(인증 서비스)가 실패할 때 마다, “security.authentication.failed” 타입의 이벤트가 발생한다. 그러면 lockout EventHandlingService 가 이벤트를 캐치하여, handleEvent(Event)메소드를 실행시킨다. 이 메소드 내부에서는 로그인 횟수를 카운트하고 있다가, 특정 값 (가령 3 회.) 이상에 도달하면, 해당 Subject 에 "lock" Credential 을 설정한다. 이후 SubjectValidationService 는 해당 Subject 에 락이 걸렸음을 확인하고, 결과적으로 이 Subject 로의 로그인 시도는 모두 실패하게 만든다. SubjectValidationService 와 EventHandlingService 를 결합하면, 한번에 3 회이상 로그인에 실패한 Subject 에 락을 걸어, 더 이상 시스템에 로그인 시도를 할 수 없게 만드는 기능을 추가할 수 있다.

그러나, EventHandlingService SPI 가 가장 빈번하게 사용되는 곳은 이벤트를 기록하는 로거를 구현할 때 이다. 이벤트는 일반 텍스트나 XML 파일에 기록되기도 하지만, 때때로 부인방지를 위해 암호화된 파일이 사용되기도 한다.

도메인당 0 개 이상의 EventHandlingService 를 설정할 수 있다.

해당 SPI 에 대한 자세한 설명은 Javadoc 과 부록 ”J”를 참고하기 바란다.

다양한 보안 SPI 클래스로부터 발생되는 표준 Event 에 대해서는 부록 ”H”를 참고하기 바란다.

9.16 Dependencies between SPI Implementations

이상에서 살펴본대로, SPI 클래스간에는 의존이 존재할 수 있다. 가령, AuthenticationService 는 AuthenticationRepositoryService 에 의존하고, LoginService 는 AuthenticationService 와 AuthorizationService 에 의존한다.

주의: 여기서 의존이란 하나의 SPI 구현 클래스가 다른 SPI 구현 클래스의 static 메소드를 호출한다는 것을 말한다(가령, AuthenticationService 구현 클래스를 AuthenticationServiceImpl 이라 하면, 이 클래스는 AuthenticationRepositoryService.getSubject(…)를 호출하여 Subject 에 대한 정보를 얻어온다).

위의 언급에서 "존재 할 수 있다"라는 말에 주목하자. 이 단어는 의존이 존재하지 않을 수도 있다는 것을 암시한다. 이는 SPI 클래스를 어떻게 구현하느냐에 달렸다. 어떤 SPI 클래스도 결코 직접적으로 다른 SPI 클래스의 메소드를 호출하지 않는다(몇 가지 사소한 예외가 있긴 있다). 일반적으로 SPI 를 구현한 클래스에서 다른 SPI 구현 클래스의 메소드를 호출함으로써 의존이 성립된다.

디폴트 보안 시스템 구현에서, SPI 구현 클래스간의 의존은 [그림 16]에서 볼 수 있다(이 그림은 단순화 시킨 것이다).

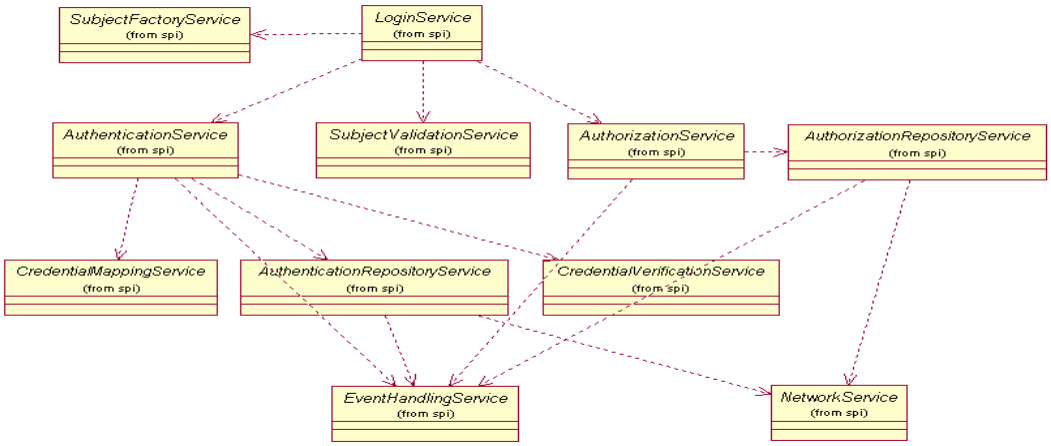

그림 18 기본 보안 시스템 구현에서의 SPI 구현 클래스

9.17 보안 서비스 설정하기

새로운 SPI 구현 클래스를 컴파일 한 다음, 이 클래스를 JEUS 보안 시스템에 등록해야 한다. 등록하는 방법은 3.4 절을 참고하기 바란다.

9.18 결론

지금까지 JEUS 보안 시스템에 새로운 보안 서비스를 구현하는 방법을 알아 보았다.

다음 장에서 커스텀 JACC 1.0 Authorization Provider 를 개발하여 JEUS 보안 시스템에 설정하는 방법에 대해 알아 보겠다.

10JACC Provider 사용

10.1 소개

본 장은 Java Authorization Contract for Containers (JACC)에 대한 핵심적인 내용만 다룬다. JACC 의 목적을 설명하고, 커스텀 JACC Provider 를 구현하여 JEUS 보안 시스템에서 사용하는 방법을 간단히 알아 보겠다.

10.2 JACC 개요

10.2.1 설명

JACC 는 “Java Authorization Contract for Containers”의 약자로, J2EE 버전 1.4 에서 처음으로 소개되었다(J2EE 1.3 과 호환된다). 현재 버전은 1.0 이다. 이는 현재 JEUS 에서 지원되는 유일한 버전이기도 하다.

JACC 배경에는 다음의 두가지 기본 목적이 있다:

- EJB 와 Servlet 권한 체크시 표준 SPI 를 제공한다.
- 기존 J2SE 보안 모델과 J2EE 보안 모델간의 조화를 추구한다.

간단히 말해서, JACC 는 EJB 와 Servlet 의 접근 권한을 정의하고 처리하는 표준적인 방법을 제시하고 있다. 따라서, JACC Provider 를 작성해 두면, JACC 호환 J2EE Server 에서 사용할 수 있다.

전체 JACC 스펙은 다음 세개의 세부 규약으로 구성되어 있다. 다음 절에서 각 세부 규약을 자세히 살펴 보겠다.

- Provider 설정 규약
- Policy 설정 규약
- Policy 결정 및 집행 규약

10.2.2 Provider 설정 규약

JACC Provider 설정 규약은 어플리케이션 서버의 런타임 환경에 JACC Provider 를 통합하는 방법을 기술하고 있다. 즉, 이 스펙은 JACC Provider 를 J2EE 서버에 등록하는 방법을 정의하고 있다. 일반적으로 이 과정은 매우 단순하다.

1. JACC Provider 는 커스텀 java.security.Policy 를 포함하고 있다.
2. "javax.security.jacc.policy.provider" 시스템 속성에 커스텀 java.security.Policy 클래스 이름을 설정한다.
3. J2EE Server 는 이 클래스명을 읽어서 인스턴스를 만든다음, java.security.Policy 타입으로 캐스팅한다.
4. 3 의 과정이 성공적으로 완료되었다면, java.security.Policy.setPolicy(…)메소드를 호출하여, 디폴트 J2SE Policy 클래스를 새로 생성한 JACC Policy 클래스로 교체한다.
5. 앞으로, 모든 권한 체크는 새로 등록된 JACC Policy 를 통해서 이루어 지고, java.security.Policy.getPolicy() 메소드를 통해 현재 JACC Policy 를 얻어올 수 있다.

주의: 위의 설명은 이해하기 쉽게 단순화한 것이다. 더 자세한 설명은 JACC 스펙을 참고하기 바란다.

10.2.3 Policy 설정 규약

Policy 설정 규약은 J2EE DD 파일 (ejb-jar.xml, web.xml)에 설정된 보안 제약 (security constraints)을 javax.security.jacc 패키지에 정의된 java.security.Permission set 으로 매핑하는 방법과 이러한 Permission 을 JACC Provider(커스텀 java.security.Policy)에 추가하는 방법을 기술하고 있다. 이는 결국 EJB 와 Servlet 에서 JACC Provider 가 권한 부여와 관련된 결정을 내릴 수 있게 만든다.

주의: Policy 설정 규약은 principal-to-role 매핑에 대한 언급이 없다. 오로지 J2EE DD 파일에 근거하여 role-to-resource 매핑을 어떻게 할 지만 정의하고 있다. 따라서 princiapl-to-role 매핑은 JACC Provider 를 공급하는 업체에 따라 각각 다른 방법으로 정의되어 있다.

간단히 말해서, Policy 설정은 다음의 과정으로 구성되어 있다.

1. JACC Provider 는 커스텀 javax.security.jacc.PolicyConfiguration Factory 를 포함하고 있다.

2. 시스템 속성 (-D 속성으로 설정) “javax.security.jacc.PolicyConfigurationFactory.provider”에 커스텀 javax.security.jacc.PolicyConfigurationFactory 클래스명을 설정한다.

3. J2EE Server 는 이 속성값을 읽어서, 해당 클래스의 인스턴스를 생성한다.(이 인스턴스를 "PCF"로 부르도록 하자).

4. Servlet 과 EJB 모듈을 deploy 한다.

5. deployment 코드는 web.xml 과 ejb-jar.xml DD 파일을 읽어 들여, 설정된 security constraints 항목을 JACC Permission 인스턴스들로 전환한다(이 Permission 클래스는 javax.security.jacc 패키지에 포함되어 있다). 이때, 각 Permission 인스턴스는 Servlet 과 EJB 에 설정되어 있는 role-to-resource 매핑을 나타낸다.

6. deployment 코드는 PCF.getPolicyConfiguration(…) 메소드를 호출하여 javax.security.jacc.PolicyConfiguration 타입의 인스턴스를 리턴 받는다(이 인스턴스를 "PC"라고 하자).

7. deployment 코드는 5 번 과정에서 생성된 role-to-resource Permission 들을 PC 의 다양한 메소드를 사용하여 PC 에 추가한다.

8. 모든 Permission 을 PC 에 추가하고 나면, PC 의 commit() 메소드를 호출한다.

9. 이제 Policy 설정은 모두 끝났다. 다음 절에서 Policy 결정과 집행 규약을 설명 하겠다.

10. Servlet 과 EJB 모듈이 undeploy 될 때, PC.delete() 메소드가 호출된다. 이는 해당 Servlet/EJB 모듈에 대한 Permission 을 제거한다.

주의: 위의 설명은 이해하기 쉽게 단순화한 것이다. 더욱 자세한 설명은 JACC 스펙을 참고하기 바란다.

10.2.4 Policy 결정 및 집행 규약

Policy 결정 및 집행 규약은 이름에서도 알수 있듯이, EJB 와 Servlet 에서 접근 권한과 관련된 결정을 어떻게 내리고 집행할 것인지 기술하고 있다. 이는 앞서 설명한 두가지 세부 규약이 모두 충족된 이후에 적용된다.

Policy 결정 및 집행은 다음과 같은 과정으로 진행된다(Servlet 을 예로 들지만, EJB 도 이와 비슷하다).

1. 해당 Servlet 페이지에 대한 요청이 들어 온다.

2. Servlet 컨테이너는 javax.security.jacc.PolicyContext 클래스를 사용하여 JACC 컨텍스트 정보를 설정한다.

3. Servlet 컨테이너는 두가지 JACC Web Permission(javax.security.jacc 패키지에 정의되어 있다) 을 생성한다. 이 Permission 인스턴스는 현재 요청한 Servlet 에 설정된 Permission 을 나타낸다.

4. Servlet 컨테이너는 Servlet 요청자가 위의 두가지 Permission 을 가지고 있는지 알아 보기 위해, JACC Provider 에 쿼리를 던진다. 쿼리를 해석하는 데는 여러가지 방법이 있다. 가령, Policy.implies(…)메소드로 확인할 수 있다. 이때 파라미터로 요청자의 Principal 로 초기화된 java.security.ProtectionDomain 에서 체크하는 모든 Permission(들)이 사용된다.

5. JACC Provider 는 2 번 과정에 세팅된 컨텍스트 정보, 3 번에서 생성한 Permission 인스턴스, 요청자의 Principal(s), principal-to-role 매핑, role-to-resource 매핑등을 모두 사용하여, 현재 요청자가 Servlet 페이지에 접근 권한을 가지고 있는지를 판단한다.

6. 권한 체크 결과가 양수값이라면, Servlet 컨테이너는 요청자가 Servlet 페이지에 접근하도록 허락한다. 그렇지 않다면, 권한 체크에 실패했다는 에러 페이지가 보여 진다.

주의: 위의 설명은 이해하기 쉽게 간단히 설명한 것이다. 자세한 설명은 JACC 스펙을 참고하기 바란다.

10.2.5 결론

본 절에서 간단하게 JACC 스펙을 알아 보았다. JACC 스펙은 세개의 세부 규약 - Provider 설정 규약, Policy 설정 규약, Policy 결정 및 집행 규약- 으로 구성되어 있다.

다음 절에서는, 커스텀 JACC Provider 를 개발하는 방법과, 개발된 Provider 를 JEUS 보안 시스템에 통합하는 방법에 대해 살펴 보겠다.

10.3 JACC Provider 개발하기

10.3.1 소개

본 절에서는, 커스텀 JACC Provider 개발과 관련된 몇 가지 지침을 소개하겠다.

10.3.2 관련 클래스

다음은 JACC Provider 를 구현하는데 필요한 클래스 목록이다.

- 추상 클래스인 `java.security.Policy` 의 구현 클래스
- 추상 클래스인 `javax.security.jacc.PolicyConfigurationFactory` 의 구현 클래스
- 인터페이스인 `javax.security.jacc.PolicyConfiguration` 의 구현 클래스

다음 절에서 각 클래스에 대해 자세히 살펴 보겠다.

10.3.3 java.security.Policy 구현하기

JACC Provider 의 핵심은 java.security.Policy 의 서브 클래스를 작성하는 것이다. 이 클래스는 JACC 에서 실제 권한 체크를 하는데 사용된다.

java.security.Policy 의 서브 클래스를 작성하기 위해 java.security.Policy 를 상속하는 새로운 클래스를 정의한다(이를 MyJACCPolicy 라고 하자). 그리고 나서, 10.2.4 절에서 설명한 Policy 결정 및 집행 규약을 구현하기 위해 implies(..)메소드를 재정의한다. 구현시, 권한 체크 질의를 해석 하기 위해, `PolicyConfiguration` 인터페이스를 통해 추가된 Permission 과 principal-to-role 매핑을 고려해야 한다.

MyJACCPolicy 는 파라미터가 없는 public 생성자를 제공해서, J2EE Server 가 인스턴스를 쉽게 생성할 수 있게 해야 한다.

JACC Policy 를 구현하는 방법에 대한 자세한 설명은 JACC 스펙의 Policy 결정 및 집행 규약과 java.security.Policy 에 대한 J2EE 1.4 Javadoc 을 참고하기 바란다.

10.3.4 javax.security.jacc.PolicyConfigurationFactory 구현하기

Policy 설정 규약을 구현하기 위해, 어플리케이션 서버는 Permission 인스턴스를 JACC java.security.Policy 에 추가할 수 있어야 한다. 이 작업은 javax.security.jacc.PolicyConfiguration 인터페이스를 통해 이루어 진다. PolicyConfiguration 인스턴스를 생성하기 위해 추상 클래스인 javax.security.jacc.PolicyConfigurationFactory 가 필요하다.

PolicyConfigurationFactory 의 서브 클래스를 작성하기 위해 javax.security.jacc.PolicyConfigurationFactory 를 상속하는 새로운 클래스를 정의한다(이를 MyJACCPolicyConfigurationFactory 라 하자). 이 클래스는 getPolicyConfiguration(…) 를 반드시 재정의 해야 한다.

MyJACCPolicyConfigurationFactory 클래스는 파라미터가 없는 public 생성자를 제공해서, J2EE Server 가 해당 클래스의 인스턴스를 쉽게 생성할 수 있도록 해야 한다.

javax.security.jacc.PolicyConfigurationFactory 를 구현하는 방법에 대한 자세한 설명은 JACC 스펙의 Policy 설정 규약 부분과 javax.security.jacc.PolicyConfigurationFactory 에 대한 J2EE 1.4 Javadoc 을 참고하기 바란다.

10.3.5 javax.security.jacc.PolicyConfiguration 인터페이스 구현하기

Policy 설정 규약을 구현하기 위해, 어플리케이션 서버는 Permission 인스턴스를 JACC java.security.Policy 에 추가할 수 있어야 한다. 이 작업은 javax.security.jacc.PolicyConfiguration 인터페이스를 통해 이루어 진다. 이 인터페이스의 인스턴스는 위에서 설명한 대로 javax.security.jacc.PolicyConfigurationFactory 를 통해 생성된다.

JACC Provider 는 javax.security.jacc.PolicyConfiguration 을 구현한 클래스를 반드시 포함하고 있어야 한다. 이 구현 클래스를 MyJACCPolicyConfiguration 라 하자.

커스텀 javax.security.jacc.PolicyConfigurationFactory 클래스인 MyJACCPolicyConfigurationFactory 는 항상 MyJACCPolicyConfiguration 인스턴스를 리턴해야 한다.

javax.security.jacc.PolicyConfiguration 을 구현하는 방법에 대한 자세한 설명은 JACC 스펙의 Policy 설정 규약과 javax.security.jacc.PolicyConfiguration 에 대한 J2EE 1.4 Javadoc 을 참고하기 바란다.

10.3.6 커스텀 JACC Proivider 클래스 다이어그램

아래는 커스텀 JACC Provider 를 구현하기 위해 필요한 클래스들과 그 들간의 관계를 단순하게 표현한 클래스 다이어그램이다.[그림 19]. 이 다이어그램에서는 클래스 이름을 "MYJACC"로 시작했지만, 어떤 이름이 와도 상관없다.

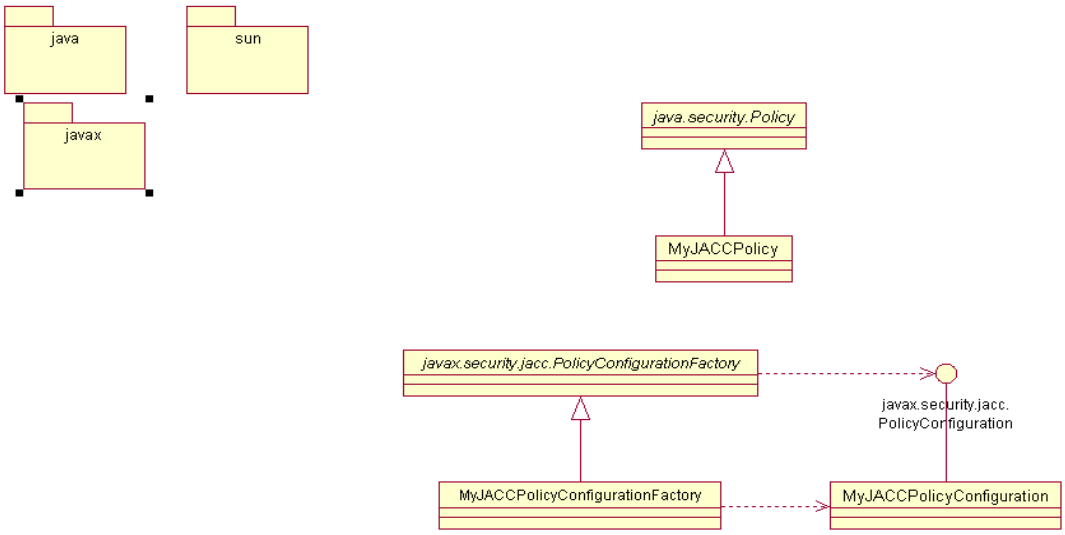

그림 19 JACC privider 클래스들

10.3.7 JACC Provider 패키징하기

보통, JACC Provider 와 Provider 가 참조하는 클래스를 JAR 로 묶어, 해당 어플리케이션 서버에 갖다 놓는다. 이를 "MYJACCProvider.jar'라고 부르도록 하자.

어플리케이션 서버의 패스에 JACC Provider jar 파일의 패스를 포함시키고, 필요한 경우, 특정 시스템 속성값을 설정한다. 이 과정은 어플리케이션 서버마다 약간씩 틀려진다. JEUS 에서 어떻게 설정하는지는 다음 절에 설명할 것이다.

10.3.8 Default JACC Provider

JEUS 보안 시스템은 매우 간단한 디폴트 JACC Provider 를 제공하고 있다. 일반적으로 이 디폴트 Provider 를 JACC 스펙을 테스트하는 이외의 용도로 사용하는 것을 권장하지 않는다. 대신, 이전 장에서 설명한 Default Authorization Provider 를 사용할 것을 권한다.

그러나, 디폴트 이외의 JACC Provider 를 사용하고자 한다면, 상용 제품은 제공되고 있지 않으므로, 자신만의 JACC JAR 아카이브를 직접 만들고, 아카이브 내에 다음 JACC Provider 파일을 포함시키도록 한다.

다음절에서 디폴트 JACC Provider 를 JEUS 에 설치하는 방법을 설펴 보겠다.

10.3.9 결론

본 절에서는 J2EE 1.4 와 1.3 서버에서 커스텀 JACC Provider 를 개발하는 방법에 대해 간단히 살펴 보았다.

다음 절에서는 JEUS 보안 시스템과 디폴트 JACC Provider 를 통합하는 방법에 대해 살펴 보겠다.

10.4 JEUS 보안 시스템과 JACC Provider 통합하기

10.4.1 소개

JEUS 보안 시스템과 JACC Provider 를 통합하기 위한 과정은 다음과 같다.

- principal-to-role 매퍼를 구현한다.
- 보안 설정 파일을 세팅한다.
- 시스템 패스에 JACC Provider JAR 패스를 추가한다.
- Java 시스템 속성을 설정한다.

10.4.2 Principal-to-Role 매퍼 구현하기

JACC 인터페이스는 principal-to-role 매핑에 대해 어떤 것도 언급하고 있지 않으므로(단지 role-to-resource 매핑만 정의하고 있다), principal-to-role 매핑을 위해, JEUS 만의 별도의 인터페이스가 필요하다.

이를 위해 JEUS 는 jeus.security.impl.aznrep.JACCPrincipalRoleMapper 라는 인터페이스를 제공한다. 이 인터페이스는 구현해야 할 단 하나의 메소드인

addPrincipalRoleMapping(PermissionMap map, String policyId)를 포함하고 있다. 이 메소드는 principal-to-role 매핑을 policyId 로 대표되는 PolicyConfiguration 에 추가한다.

주의: principal-to-role 매핑은 J2EE 에서 어플리케이션 범위를 가진다는 점을 명심하기 바란다. 왜냐하면, 같은 J2EE 어플리케이션에 속한 모든 principal-to-role 매핑은 하나의 Map 에 병합되기 때문이다. PermissionMap 클래스와 PermissionMap 인스턴스를 서로 병합하는데 사용되는 "add"메소드에 대한 자세한 정보는 PermissionMap 클래스에 대한 API 문서를 참고하기 바란다.

JACCPrincipalRoleMapper 인터페이스를 구현한 클래스는 파라미터가 없는 public 생성자를 제공해야 한다. 그리고 반드시 JACC Provider JAR 내에 추가되어야 한다.

JACCPrincipalRoleMapper 가 pricinpal-to-role 매핑을 생성하는 과정은 다음과 같다. 먼저 jeus.security.jacc.principalRoleMapper 시스템 속성으로 jeus.security.jacc.principalRoleMapper 를 구현한 클래스명을 설정한다. 그러면, jeus.security.impl.aznrep.JACCAuthorizationRepositoryService 를 구현한 클래스가 이 속성값을 읽어, Class.forName(mapperClassname).newInstance() 메소드로 해당 인스턴스를 생성한다. 이 후, 생성된 인스턴스의 addPrincipalRoleMapping() 메소드를 호출하여 JEUS DD 파일에 명시된 principal-to-role 매핑을 생성해서 추가한다.

이 인터페이스에 대한 자세한 정보는 부록 J 와 Javadoc 을 참고하기 바란다.

10.4.3 JACC 보안 설정 파일 세팅하기

JEUS 보안 시스템은 두개의 어답터 클래스를 사용하여 JEUS native authorization API 와 JACC authorization API 를 연결한다. 이 클래스는 다음과 같다.

- jeus.security.impl.azn.JACCAuthorizationService
- jeus.security.impl.aznrep.JACCAuthorizationRepositoryService

전자는 컨테이너에서 Policy 결정 및 집행 규약을 구현하는 부분으로, 권한을 체크하기 위해 java.security.Policy.getPolicy().implies(. . .)메소드를 호출한다.

후자는 Policy 설정 규약을 구현한 부분으로, 컨테이너가 생성한 jeus.security.base.Policy 인스턴스를 JACC Provider 의 구성 요소인 PolicyConfiguration 인스턴스에 추가하는 역할을 한다.

JACC 를 작동시키기 위해서는, 이 두가지 보안 서비스가 master.xml 과 slave.xml 에 설정되어 있어야 한다.

예제 :

<< *master.xml and slave.xml.*>>

```
<?xml version="1.0"?>

<security-domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <name>JACC_DOMAIN</name>
    . . .
    <security-service>
        <classname>
            jeus.security.impl.azn.JACCAuthorizationService
        </classname>
    </security-service>
    <security-service>
        <classname>
     jeus.security.impl.aznrep.JACCAuthorizationRepositoryService
        </classname>
    </security-service>
    . . .
</security-domain>
```

10.4.4 시스템 패스에 JACC Provider JAR 추가하기

시스템 패스에 JACC provider JAR 파일을 추가하기 위해, JEUS_HOME\lib\system 디렉토리에 JAR 파일을 갖다 놓는다.

10.4.5 JACC 에 대한 Java 시스템 속성 설정하기

JACC 규약과 JEUS 는 어플리케이션 서버가 JACC Provider 를 인식하도록, 다음 세가지 Java 시스템 속성을 규정하고 있다.

- javax.security.jacc.policy.provider: JACC Provider 를 나타내며, java.security.Policy 를 구현한 클래스명.

- javax.security.jacc.PolicyConfigurationFactory.provider: PolicyConfiguration 인스턴스를 생성하고 로딩하는 PolicyConfigurationFactory 를 구현한 클래스명.

- jeus.security.jacc.principalRoleMapper:
  jeus.security.impl.aznrep.JACCPrincipalRoleMapper 인터페이스를 구현한 클래스명으로 JEUS DD 파일로부터 principal-to-role 매핑을 만들어 낸다.

이 세가지 시스템 속성은 JEUS_HOME\bin\jeus 스크립트에 추가되어야 한다. 또한 모든 엔진 컨테이너에 대해서 JEUSMain.xml 에 <common-option>으로 설정되어야 한다.

예제:

<<jeus.cmd 또는 jeus>>

```
. . .
java -server . . .
      -Djavax.security.jacc.policy.provider=
      myprovider.MyJACCPolicy
      –Djavax.security.jacc.PolicyConfigurationFactory.provider=
      myprovider.MyJACCPolicyConfigurationFactory
      –Djeus.security.jacc.principalRoleMapper=
      myprovider.MyJACCPrincipalToRoleMapper
      jeus.server.JeusBootstrapper %*
```

<<JEUSMain.xml>>

```
<?xml version="1.0"?>
<jeus-system xmlns=“http://www.tmaxsoft.com/xml/ns/jeus”>
    <node>
        <name>@HOSTNAME@</name>
            <engine-container>
            <name>container1</name>
            <!-- engine container JVM option -->
            <command-option>
            . . .
       -Djavax.security.jacc.policy.provider=
       myprovider.MyJACCPolicy
       –Djavax.security.jacc.PolicyConfigurationFactory.provider=
       myprovider.MyJACCPolicyConfigurationFactory
       –Djeus.security.jacc.principalRoleMapper=
       myprovider.MyJACCPrincipalToRoleMapper
            </command-option>
```

```
            <engine-command>
                <type>servlet</type>
                <name>engine1</name>
                <startup-mode>xml</startup-mode>
            </engine-command>
        . . .
```

10.4.6 디폴트 JACC Provider 클래스명

이전에 언급한 대로, 디폴트 JACC provider 클래스명은 다음과 같다.

- Policy:
  jeus.security.impl.jacc.JACCPolicyWrapper
- PolicyConfigurationFactory:
  jeus.security.impl.jacc.JACCPolicyConfigurationFactoryImpl
- JACCPrincipalRoleMapper:
  jeus.security.impl.jacc.JACCDefaultPrincipalRoleMapper

이 클래스들은 모두 JEUS_HOME\lib\system\jeus.jar 내에 패키징 되어 있다.

10.4.7 결론

지금까지 JEUS 보안 시스템에 JACC Provider 를 통합하는 방법을 알아 보았다. 이 과정은 크게 세가지 단계 - 보안 설정 파일 세팅하기, JACC Provider Jar 파일을 시스템 패스에 설정하기, JEUS 스크립트와 JEUSMain.xml 에 시스템 속성값 설정하기 - 로 구성되어 있다.

마지막으로, 디폴트 JACC Provider 클래스 이름을 알아 보았다.

10.5 결론

지금까지 JEUS 에서 JACC 1.0 스펙을 적용하는 방법에 대해 알아 보았다. 우선 JACC 스펙이 무엇인지 알아 보았고 (세 가지 하위 규약), JACC Provider 를 구현하여 패키징 해 보았다. 그리고 패키징한 JACC Provider 를 JEUS 에 통합해 보았다(principal-to-role 매핑을 구현하고, 보안 설정 파일을 세팅하고, 관련 시스템 속성을 세팅하였다).

JACC 스펙에 대한 자세한 정보는 JACC 1.0 스펙 문서를 참고하기 바란다. 해당 문서는 http://java.sun.com/j2ee 에서 찾아 볼 수 있다.

11결론

지금까지 JEUS Security 서비스를 살펴보았다. JEUS 의 Security 서비스는 기본적으로 SPI 에 근거하여 서비스되고 있으며, SPI 를 상속한 여러 Custom Service 들이 JEUS 의 서비스로 등록될 수 있다. 또한 JEUS5 에 JACC 1.0 스펙을 적용하는 방법들을 살펴보았다.

이와 같이 JEUS Security Service 를 적절히 이용하여서 JEUS 내에서 관리하는 resource 들을 관리하고 통제할 수 있다.

A SecurityAdmin 툴 사용

A.1 소개

SecurityAdmin 툴은 JEUS 보안과 관련된 기본적인 사항을 관리해 주는 커맨드 라인 툴이다. 기본적으로 Subject 와 Policy 를 추가하거나 삭제하는 데 사용된다.

SecurityAdmin 툴은 유연하고 정교하게 설계되어 있어서, 프롬프트 인터페이스로 사용자와의 상호 작용이 가능할 뿐 아니라, 스크립트 파일을 사용하여 배치 작업도 처리할 수 있게 해 준다. 또한 "assertion" 명령어는 보안 시스템이 제대로 동작하고 있는지 검증해 준다.

SecurityAdmin 툴은 원격지 보안 서버와 통신할 때 항상 SSL 연결을 사용한다.

A.2 호출 방법

SecurityAdmin 툴을 시작하기 위해서는, 커맨드 창에서 윈도우는 securityadmin.cmd 를 유닉스의 경우는 seurityadmin 을 실행한다. 해당 스크립트는 JEUS_HOME/bin 디렉토리에 있다.

호출 문법은 다음과 같다.

```
securityadmin
      –help | ? |
      -version |
      ([-domain <domain name>]
       [-host <JEUS manager host name>]
       [-port <JEUS manager security port>]
       [-silent]
       [-noprompt]
       [-commandfile <filename of text file containing commands>]
       [-commands “<command 1>” “<command 2>” “<command N>”...])
```

각 옵션에 대한 설명은 다음과 같다.

`-help | ?`

호출 문법과 함께 도움말을 보여준다.

`-version`

버전 정보를 보여준다.

`-domain <domain name>`

대상 보안 도메인을 설정한다. 만약 설정되어 있지 않다면, "SYSTEM_DOMAIN"으로 설정된다.

`-host <JEUS manager host name>`

연결하려는 JEUS 가 동작하고 있는 호스트의 이름을 설정한다. 디폴트는 "localhost"이다.

`-port <JEUS manager security port>`

연결하려는 JEUS 의 security 포트를 설정한다. 디폴트는 "9742"이다.

`-silent`

툴이 실행중일 때 어떤 메시지도 보여지지 않도록 설정한다.
아래에서 살펴 보겠지만 이 옵션은 스크립트 파일을 사용할 때 유용하다.

-noprompt

툴이 시작된 이후에는 어떤 프롬프트도 보여지지 않게 하는 것이다. 모든 설정 사항은 -commandfile, -commands 옵션으로 미리 주어진다.

`-commandfile <filename of text file containing commands>`

특정 파일에 설정된 명령어들을 읽어 들여, 툴을 시작하게 하는 것이다. 파일 내의 모든 명령어는 프롬프트 창을 열어 툴을 작동시킬 때와 문법이 동일해야 한다. 자세한 사항은 다음 절에서 확인할 수 있다.

`-commands “<command 1>” “<command 2>” “<command N>”...`

-commands 옵션은 여러 개의 명령어를 한 라인에 쓰는 것이다. 모든 명령어는 ""로 감싸져야 하고, 공백 문자로 분리 되어야 한다. 만약, -commandfile 옵션과 같이 사용되었을 경우, -commands 에 설정된 명령

어부터 실행된 다음, -commandfile 에 설정된 명령어가 실행된다. 이 옵션은 반드시 옵션들 중 가장 마지막에 위치해야 한다.

예제:

```
securityadmin –host webapps–port 10000 –domain SYSTEM_DOMAIN
–silent –noprompt –commandfile .\mysecurityscript.txt –
commands “login –username johan –password johanpw”
```

위의 명령어는 호스트 이름이 "webapps"이고, security 포트가 10000 인 JEUS 에 커넥션을 맺는다. 타겟 도메인은 "SYSTEM_DOMAIN"이다. 툴은 어떤 메시지도 출력하지 않고, 프롬프트도 보여주지 않는다. 대신 시작하자 마자 “login –username johan –password johanpw"를 실행하고, 이후 “.\mysecurityscript.txt”에 언급되어 있는 명령어들을 차례로 실행한 다음, 종료한다.

```
securityadmin
```

위의 명령어는 "localhost"에 기본 security 포트 번호인 "9742"로 JEUS 에 커넥션을 맺는다. 그리고, 프롬프트를 보여주면서, 대화식으로 명령어를 직접 입력해 가면서 툴을 작동시킨다. 타겟 도메인은 "SYSTEM_DOMAIN"이다. 다음절에서 프롬프트에 어떤 명령어를 어떻게 입력해야 하는지 자세히 설명할 것이다.

A.3 명령어

A.3.1 소개

본 절은 SecurityAdmin 툴에서 사용하는 명령어에 대해 소개 하겠다. 명령어는 툴을 시작하기 전에 명령어 라인 파라미터로 입력할 수도 있고, 프롬프트에서 "?"기호 다음에 직접 입력할 수도 있다.

대부분의 기본 명령어는 다음과 같은 구조를 가진다.

```
<command name>
      -<argument name> <argument value>
      -<argument 2 name> <argument 2 value>
      . . .
```

모든 명령어는 유일한 "command name"으로 시작하고 0 개 이상의 name-value 파라미터 쌍을 가지고 있다. 이때 파라미터명은 "-"로 시작하고, name 과 value 는 공백문자로 구분된다. 모든 명령어는 한 라인에 작성되어야 하고, value 에

공백 문자가 있을 경우 ""으로 감싸준다. 아래는 두 가지 예제를 보여주고 있다. ("?"는 프롬프트이다.)

```
? login –username johan –password johanpw
? setsubjectdescription –username johan –description “A
really nice guy!”
```

주의: 아래 절에서 살펴 보겠지만, 기본 명령어 중에 이런 구조를 갖지 않는 것도 있다.

A.3.2 기본 명령어

기본 명령어는 툴 자체를 관리하기 위해 사용되는 내장 명령어로써 다음과 같은 것들이 있다.

`help`

모든 명령어에 대한 간단한 도움말을 보여준다.

help <command name>

"help login"과 같이 사용되며, 특정 명령어에 대한 도움말을 보여준다.

주의: 이 명령어는 기본 명령어들과는 함께 사용할 수 없다. 즉 "hep help" 이런 식으로 사용할 수 없다.

`asserttrue <command>`

해당 명령어가 제대로 동작하는지 검증한다. 해당 명령어 실행 중에 문제가 발생하면, "assert failed"메시지가 출력된다. 만약 문제가 발생하지 않으면, 어떤 메시지도 출력하지 않은채 종료된다. 이 명령어는 보안 시스템을 테스팅하는데 사용할 수 있으며, 주로 스크립트 파일내에서 사용된다.

예제:

```
? asserttrue login –username johan –password johanpw
```

`assertfalse <command>`

해당 명령어가 제대로 동작 안하는지 검증한다. 해당 명령어 실행 중에 아무 문제도 발생하지 않으면, "assert failed"메시지가 출력된다. 만약 문제가 발생한다면, 어떤 메시지도 출력하지 않은 채 종료된다. 이 명령

어는 보안 시스템을 테스팅하는데 사용할 수 있으며, 주로 스크립트 파일내에서 사용된다.

예제 :

```
? assertfalse login –username johan –password johanpw
```

`echo <text>`

해당 텍스트를 출력한다. 스크립트 파일내에서 유용하다.

`# <text>`

해당 텍스트를 주석처리 한다. 스크립트 파일내에서 유용하다.

`exit`

SecurityAdmin 툴을 종료하고, 원격지 JEUS 와의 커넥션을 끊는다.

A.3.3 로그인과 로그아웃 명령어

로그인과 로그아웃 명령어는 로컬에서 특정 Subject(보통 administrator)로 로그인해서 SecurityAdmin 의 다른 명령어들을 호출할 수 있게 해준다. 로그인된 Subject 는 원격지 호출이 발생할 때 마다, 원격지 JEUS 에 전파된다.

```
login –username <user name>
      -password <plain password>
      [-domain <domain name>]
```

기본적인 패스워드 베이스 메커니즘을 사용하여, 특정 사용자로 로그인을 시도한다. 만약 "login"이 별도로 호출되지 않았다면, "anonymous"로 로그인된 걸로 간주한다. "login"은 서로 다른 Subject 로 여러 번 호출될 수 있으며, 해당 Subject 는 로그인 스택에 차곡차곡 쌓인다. 새로운 "login"은 기존의 로그인된 Subject 를 강제로 로그아웃 시키지 않고, 단순히 이전에 로그인된 Subject 를 감추고 있다가, 로그아웃되면 이전의 Subject 가 다시 활성화된다. 만약 "domain"파라미터가 지정되어 있다면, 해당 도메인은 디폴트 도메인 대신에 사용된다.

`login2`

커스텀 로그인 메커니즘을 사용하여, 특정 사용자로 로그인을 시도한다. 이 명령어는 로그인 과정을 별도로 설정된 로그인 메커니즘에 맡기고 있으며, 반드시 패스워드 기반의 메커니즘이 사용된다고 가정할 수

없다. 로그인 커스터마이징과 관련된 자세한 사항은 9.12 절을 참고하기 바란다.

`Logincode [-domain <domain name>]`

어떠한 인증 서비스도 거치지 않고, 모든 권한을 가지는 특별한 "code Subject"로 로그인한다. 이를 설정하기 위해서는 반드시 원격지 서버 JVM 의 전역 시스템 패스워드와 동일한 패스워드를 현재 JVM 의 전역 시스템 패스워드로 설정해 주어야 한다. 이 설정은 securityadmin 부트 스크립트와 jeus 부트 스크립트에 `-Djeus.security.globalPassword` 속성값으로 주어진다. 더 자세한 정보는 본 부록 I, 3 과 3.7.1 절을 참고하기바란다. "code Subject"는 5.4 절에서 설명 했듯이 초기화 되지 않은 새로운 보안 도메인을 부팅할 때 사용된다. 일반적인 상황에서 이 명령어는 결코 사용되어서는 안된다. -domain 파라미터를 사용해서 로그인할 도메인을 설정할 수 있다. 설정된 도메인은 securityadmin 스크립트에 포함된 -domain 파라미터이거나 SYSTEM_DOMAIN 일 수 있다.

`currentdomain`

현재 로그인된 Subject 의 보안 도메인을 보여준다.

`currentsubjectname`

현재 로그인된 Subject 명을 보여준다.

`currentsubject`

현재 로그인된 Subject 를 보여준다.

`logout`

현재 로그인된 Subject 를 로그아웃하고, 이전에 로그인된 Subject 를 활성화 시킨다. "anonymous" Subject 로는 로그아웃 할 수 없다.

A.3.4 Subject 관리 명령어

Subject 관리 명령어는 Subject 를 추가, 수정, 삭제하는데 사용된다.

주의: 본 절에서 소개하는 명령어는 매우 민감한 사안을 다루므로, 이 명령어를 실행하기에 적절한 권한을 가진 Subject 로 로그인해야 한다. SecurityAdmin 에 특정 Subject 로 로그인과 로그아웃하는 것과 관련된 사항은 “Login & Logout"절을 참고하기 바란다.

`addsubject -username <user name>`

보안 시스템에 새로운 Subject 를 추가한다. 주어진 사용자명은 현재 도메인내에서 유일해야 한다("getsubjectnames"명령어로 확인할 수 있다). 사용자명은 일련의 호출 과정에서 특정 Subject 를 참조 하기 위해 사용된다. 즉 Subject 대한 id 에 해당된다.

```
removesubject -username <user name>
```

현재 도메인에서 이미 존재하고 있는 Subject 를 삭제한다.

```
getsubjectnames
```

현재 도메인에서 모든 사용자명(Subject 명) 리스트를 보여준다.

```
getsubject -username <user name>
```

주어진 Subject 에 대한 자세한 정보를 보여준다.

```
setpassword -username <user name> -password <clear text password>
```

주어진 Subject 에 대한 패스워드를 설정한다. 패스워드는 어떤 암호화나 인코딩없는 일반 문자로 주어져야 한다.

경고: 이 명령어는 해당 Subject 에 이전에 주어졌던 모든 패스워드를 지워버린다.

```
locksubject -username <user name>
```

이 명령어는 특정 Subject 에 잠금을 건다. “lockout validation service”가 서버에 제대로 구성되어 있다면, 해당 Subject 에 대한 모든 로그인 시도는 "unlocksubject"가 호출되기 전에 모두 실패로 끝난다. “lockout validation service”를 설정하는 방법은 4.2 절을 참고하기 바란다.

```
unlocksubject -username <user name>
```

“locksubject”.로 걸었던 잠금을 해제한다.

```
sesubjectexpiration -username <user name>
        -expirytime <YYYY MM DD HH MM>
```

Subject 에 대한 만료시간을 설정한다. 만료일자 파라미터 포맷은 YYYY MM DD HH MM (example: “2007 01 01 19 30”) 이다. “expiration validation service”가 제대로 설정되어 있다면, 만료 시간 이후에 주어진 Subject 로의 로그인은 모두 실패로 돌아간다. “expiration validation service”를 설정하는 자세한 정보는 4.3 절을 참고하기 바란다.

```
clearsubjectexpiration -username <user name>
```

"setsubjectexpiration" 명령어로 설정한 Subject 의 만료시간을 해제한다.

```
addprincipal -username <user name> -principal <principal name>
```

특정 Subject 에 새로운 Principal 을 설정한다(선택 사항이다). 새로운 Principal 은 Subject 에 대한 user name 일 수도 있고, 만약 하나 이상의 Subject 에 동일한 Principal 이 부여되어 있다면, Subject group 명 일 수도 있다.

```
removeprincipal -username <user name> -principal <principal name>
```

Subject 로부터 Principal 을 삭제한다. "addprincipal"로 추가된 Principal 을 삭제하는데 사용할 수 있다.

```
setsubjectdescription -username <user name> -description "<text>"
```

사용자명이 나타내는 Subject 에 대한 설명을 추가한다. 만약 설명문 내에 공백이 포함되어 있다면 ""로 감싸 주어야 한다.

```
addcredential -username <user name>
      -classname <CredentialFactory classname>
      [-properties <name>=<value1>;<name2>=<value2>;...]
```

사용자명이 나타내는 Subject 에 커스텀 Credential 을 추가한다. 실제 CredentialFactory 인터페이스를 구현한 클래스명을 지정하는데, 이 인터페이스는 Credential 을 생성하는데 사용된다. -properties 옵션을 사용해서, CredentialFactory 에 필요한 속성을 제공할 수 있다. 예제 :

```
? addcredential -username javajoe
      –classname jeus.security.resource.PasswordFactory
      –properties password=pw
```

위의 명령어는 "javajoe"라는 사용자에 CredentialFactory 로 PasswordFactory 를 설정한 예이다. "javajoe"의 패스워드는 "pw"이다.

```
removecredential -username <user name>
      -classname <CredentialFactory classname>
      [-properties <name>=<value1>;<name2>=<value2>;...]
```

Subject 로부터 CredentialFactory 를 삭제한다. 이 명령어는 "addcredential"로 설정된 Credential 을 삭제한다.

A.3.5 Policy 관리 명령어

Policy 명령어는 보안 시스템에서 Policy 와 관련된 사항을 관리하는데 사용된다. 이 명령어를 통해서 principal-role 매핑, role-resource 매핑을 추가하거나 삭제할 수도 있다. Policy 명령어는 보안 시스템에서 권한 체크와 관련된 전반적인 사항을 관리할 수 있다.

주의: 이 명령어는 매우 민감한 사안을 다루기 대문에, 반드시 적절한 권한을 가진 Subject 로 로그인해야 한다. SecurityAdmin 에서 특정 Subject 로 로그인, 로그아웃 하는 방법은 “Login & Logout”절을 참고하기 바란다.

```
getpolicy [-contextid <context id>]
```

특정 context id 를 포함하고 있는 Policy 를 보여준다. 만약 -context 가 생략되어 있다면, "default"가 사용된다.

```
getpolicyids
```

보안 시스템에서 설정되어 있는 id 리스트를 보여준다.

```
assignrole –principal <principal name> -role <role name>
            [-actions <actions for role>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Role 을 Principal 에 할당한다. -actions 파라미터는 -classname 으로 설정한 클래스에 두 번째 생성자 파라미터로 "actions"를 넘길 때 사용한다. -classname 으로 설정한 클래스는 java.security.Permission 추상 클래스를 확장한 Java 클래스여야 하며 최소한 하나의 String 타입의 "role"을 받아 들이는 생성자를 가지고 있어야 한다. 디폴트 클래스는 jeus.security.resource.RolePermission 이고, 이 클래스는 생성자에서 "actions"가 아닌 "role"만을 받아 들인다. contextid 는 "default"이외의 다른 context 를 지정할 때 사용된다.

```
unassignrole –principal <principal name> -role <role name>
            [-actions <actions for role>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Principal 에서 Role 을 삭제한다.

```
excluderole -role <role name>
            [-actions <comma-separated list of actions>]
```

```
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Role 을 배제한다(해당 Role 은 모두에게 거부된다).

```
includerole -role <role name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

"excluderole" 설정을 해제한다.

```
uncheckrole -role <role name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

role 을 체크하지 않는다(해당 role 은 모두에게 허용된다).

```
checkrole -role <role name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

"uncheckrole" 설정을 해제한다.

```
assignresource –role <role name> -resource <resource name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Role 에 특정 Resource 와 Action 을 할당한다.

Resource 이름은 보통 "jeus.server"와 같은 Resource 를 나타내는 Java 클래스 명이다. 그리고, actions 는 Resource 에 대한 Action 목록이며, 각 Action 은 공백으로 구분된다. 가령, jeus.serverresource 의 경우 "boot", "down"과 같은 Action 이 있다. Resource Permission 에 대한 더 자세한 정보는 jeus.security.resource.ResourcePermission 클래스를 참고하기 바란다. 그리고, JEUS 가 어떤 Resource Permission 을 체크하는지에 관한 정보는 본 부록 ”G”를 참고하기 바란다.

-actions 파라미터는 -classname 으로 설정한 클래스에 두 번째 생성자 파라미터로 "actions"를 넘길 때 사용한다. -classname 으로 설정한 클래스

는 java.security.Permission 추상 클래스를 확장한 Java 클래스여야 하며 최소한 하나의 String 타입의 "resource"을 받아 들이는 생성자를 가지고 있어야 한다. 디폴트 클래스명은 jeus.security.resource.ResourcePermission 이고, 이는 "resouce"와 "actions"를 생성자 파라미터로 차례로 받아 들인다. contextid 는 "default"이외의 context 를 지정할 때 사용된다.

```
unassignresource –role <role name> -resource <resource name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Role 로부터 Resource 를 삭제할 때 사용한다(Role 은 더 이상 해당 Resource 에 접근할 수 없다).

```
excluderesource -resource <resource name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Resource 를 배제한다(resource 는 모두에게 거부된다).

```
includeresource -resource <resource name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

“excluderesource”로 설정된 사항을 해제한다.

```
uncheckresource -resource <resource name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

Resource 체크를 해제한다(resource 는 누구에게나 허용된다).

```
checkresource -resource <resource name>
            [-actions <comma-separated list of actions>]
            [-contextid <context id>]
            [-classname <Permission class name>]
```

“uncheckresource”로 설정된 사항을 해제한다.

A.3.6 권한 체크 명령어

권한 체크 명령어는 현재 로그인된 Subject 가 특정 권한을 가지고 있는지 체크할 때 사용된다. 이는 구체적으로 Subject 가 특정 Role 에 포함되어 있는지, 또는 Subject 가 특정 Resource 에 접근해서 특정 Action 을 실행할 수 있는지 를 체크하는 것이다. 권한 체크 명령어의 결과는 현재 적용된 Policy 에 따라 달라진다. Policy 정보는 이전 절에 설명한 명령어들로 변경할 수 있다.

```
authorizerole -role <role name>
              [-contextid <context id>]
```

현재 로그인된 Subject 가 특정 Role 에 포함되어 있는지 체크한다. -contextid 는 선택 사항으로 "default"이외의 Context 를 지정하고자 할 때 사용된다.

```
authorizeresource -resource <resource name>
            [-action <action>]
            [-contextid <context id>]
```

현재 로그인된 Subject 가 특정 Resource 와 Action 에 권한을 가지고 있는지 체크한다. -contextid 는 선택 사항으로 "default"이외의 Context 를 지정하고자 할 때 사용된다.

A.4 Sample SecurityAdmin Script File

이전 장에서 언급했듯이, SecurityAdmin 툴은 명령어가 포함되어 있는 스크립트로 시작할 수 있다. 아래는 스크립트 파일의 예제이다(이 예제에는 보안 시스템의 다양한 명령어들을 테스트하기 위하여 "asserttrue"와 "assertfalse" 명령어가 사용되었다).

<<*testscript.txt*>>

```
echo Running tests...
assertfalse login -username haha -password haha
asserttrue login -username johan -password johanpw
asserttrue addsubject -username pelle
asserttrue addprincipal -username pelle -principal apa
assertfalse login -username pelle -password pellepw
asserttrue setpassword -username pelle -password pellepw
asserttrue login -username pelle -password pellepw
asserttrue logout
asserttrue assignrole -principal apa -role role1
```

```
asserttrue assignresource -role role1 -resource rsc -actions 1,2,3
asserttrue login -username pelle -password pellepw
asserttrue authorizerole -role role1
asserttrue authorizeresource -resource rsc -action 2
asserttrue logout
asserttrue unassignrole -principal apa -role role1
asserttrue unassignresource -role role1 -resource rsc -actions 1,2,
3
asserttrue assignrole -principal apa -role role1 -contextid myconte
xt
asserttrue assignresource -role role1 -resource rsc -actions 1,2,3
 -contextid mycontext
asserttrue login -username pelle -password pellepw
assertfalse authorizerole -role role1
assertfalse authorizeresource -resource rsc -action 2
asserttrue authorizerole -role role1 -contextid mycontext
asserttrue authorizeresource -resource rsc -action 2 -contextid myc
ontext
asserttrue logout
asserttrue unassignrole -principal apa -role role1 -contextid mycon
text
asserttrue unassignresource -role role1 -resource rsc -actions 1,2,
3 -contextid mycontext
asserttrue login -username pelle -password pellepw
assertfalse authorizerole -role role1
assertfalse authorizeresource -resource rsc -action 2
assertfalse authorizerole -role role1 -contextid mycontext
assertfalse authorizeresource -resource rsc -action 2 -contextid my
context
asserttrue logout
asserttrue removesubject -username pelle
assertfalse login -username pelle -password pellepw
asserttrue logout
echo Test run finished.
echo If you did not see any "assert failed" messages, all is well!
echo If errors occurred, make sure to run with a clean configuratio
n
echo that does not have the Subject "pelle".
```

주의: 위의 예제는 여러 라인에 걸쳐 쓰여져 있는 것 같지만, 실제로 모든 명령어는 하나의 라인에 쓰여 져야 한다.

주의 2: "asserttrue"와 "assertfalse" 명령어는 검증이 주요 관심사가 아니라면 생략할 수 있다.

위의 스크립트로 SecurityAdmin 을 실행하고자 한다면, 명령어 라인에 다음과 같이 입력하도록 한다. 이때 스크립트 파일은 워킹 디렉토리에 있고, JEUS 는 localhost 에서 동작중이라고 가정한다.

```
securityadmin –noprompt –commandfile testscript.txt
```

B base64 커맨드 라인 툴 레퍼런스

B.1 소개

base64 커맨드 라인 툴은 JEUS_HOME\bin\base64 에 있으며, 텍스트를 Base64 포맷으로 인코딩하고, Base64 포맷으로 인코딩된 텍스트를 디코딩하는데 사용된다. 보통 이 툴은 subjects.xml 에 쓰여지는 패스워드를 인코딩할 때 사용된다.

중요사항: Base 64 포맷은 인코딩 알고리즘이지, 암호 알고리즘이 아니다. 따라서 패스워드와 같은 일반 텍스트를 Base 64 포맷으로 변환한 후에 다시 일반 텍스트로 복구하는 것은 매우 간단한 일이다. 그럼에도 subjects.xml 에 Base 64 를 사용하는 것은 administrator 를 제외한 다른 계정의 사용자가 실수로 subjects.xml 에 저장되어 있는 패스워드를 보지 못하게 하기 위해서다. 또다른 이유는 XML 파일과 database 필드에 non-standard character 를 사용하지 못하도록 하기 위해서다(Base 64 는 “standard” ASCII character 만을 사용한다).

B.2 호출 방법

Base 64 툴은 다음과 같이 호출된다. 이때 JEUS_HOME 시스템 환경변수가 설정되어 있고, 시스템 패스에 JEUS_HOME\bin 디렉토리가 설정되어 있다고 가정한다.

호출 후에, 툴은 단순히 인코딩된 또는 디코딩된 문자열을 보여준다. 만약 툴에 어떤 파라미터도 제공하지 않았거나, 호출 문법이 잘못되었을 경우, 단순한 도움말 메시지가 보여진다.

예제:

```
C:\> base64 encode mypassword
bXlwYXNzd29yZA==

C:\> base64 decode bXlwYXNzd29yZA==
mypassword
```

문자열 “bXlwYXNzd29yZA=="는 "mypassword"를 Base 64 포맷으로 인코딩한 것이다. 이것은 subjects.xml 에 "password" 속성값으로 저장된다.

C master.xml 과 slave.xml

C.1 소개

본 레퍼런스 부록은 보안 시스템의 주요 설정 파일인 "master.xml"과 "slave.xml"의 모든 XML 엘리먼트를 설명한다. 해당 "security.xsd"파일은 JEUS_HOME\config\xsds 디렉토리에 있다.

본 부록은 다음의 3 가지 하위 절로 구성되어 있다.

1. **XML SCHEMA/XML tree 는**, XML 구성 파일의 모든 엘리먼트의 계층적인 목록이다. 각 엘리먼트 트리의 형식은 다음과 같다.

    a. **index number** (예 (11))는, 태그 레퍼런스로 각 태그마다 인덱스를 붙여 놓은 것이다. 태그 레퍼런스는 이 번호순서대로 설명한다.

    b. XML SCHEMA 에서 정의한 XML 태그 명을 <element name>형식으로 표시한다.

    c. XML SCHEMA 에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element.

    d. 몇몇 태그에는 “P”문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝 할 때 사용된다.

2. **element 레퍼런스** 는 각 XML element 마다 하나의 테이블로 구성되어 있다. 이 테이블은 다음과 같은 세부 항목들을 포함하고 있다.

    a. **Description**: XML 태그에 대한 간단한 설명

    b. **Value Description**: XML 태그에 입력하는 값과 타입

    c. **Value Type**: 값의 데이터 타입

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위한 추천 값

h. **Child Elements**: 자신의 태그 안에 다시 태그를 포함하는 경우에 사용 가능한 태그

3. **sample XML file**: 구체적인 "master.xml"의 예제.

C.2 XML SCHEMA/XML Tree

```
(1) <security-services>
      (2) <security-service>*
            (3) <classname>
            (4) <property>*
                  (5) <name>
                  (6) <value>?
```

C.3 Element Reference

(1) **<security-services>**	
Description	보안 시스템에 사용될 보안 서비스들을 정의한다.
Child Elements	(2)security-service*

(2) <security-services> **<security-service>**	
Description	보안 서비스를 정의한다.
Child Elements	(3)classname (4)property*

(3) <security-services> <security-service> **<classname>**	
Description	jeus.security.spi 의 클래스들 중 하나를 확장한 Java 클래스명

Value Description	Java 클래스명
Value Type	token
Example	`<classname>mypackage.MyAutenticationService</classname>`

(4) <security-services> <security-service> **<property>**

Description	선택사항으로 해당 보안 서비스에 name-value 쌍으로 속성을 정의한다. 정의할 수 있는 속성과 각 속성에 대한 설명은 서비스 클래스에 대한 문서를 참조하기 바란다.
Child Elements	(5)name (6)value?

(5) <security-services> <security-service> <property> **<name>**

Description	속성명
Value Description	속성명
Value Type	token
Example	`<name>user-data-file</name>`

(6) <security-services> <security-service> <property> **<value>**

Description	선택사항으로 속성에 대한 값.
Value Description	속성 값
Value Type	token
Example	`<value>c:\security\myUsers.xml</value>`

C.4 Sample master.xml File

<master.xml>>

```
<security-services>
    <security-service>
        <classname>mypackage.MyAutenticationService</classname>
        <property>
            <name>user-data-file</name>
```

```
            <value>c:\security\myUsers.xml</value>
        </property>
    </security-service>
</security-services>
```

D subjects.xml

D.1 소개

본 레퍼런스 부록은 subjects.xml 의 모든 XML 태그 에 대해 설명한다. subject.xml 은 디폴트 보안 시스템에서 user 를 정의하는데 사용된다. 해당 "subjects.xsd"파일은 JEUS_HOME\config\xsds 디렉토리에 있다.

본 부록은 다음의 3 가지 하위 절로 구성되어 있다.

1. **XML SCHEMA/XML tree 는**, XML 구성 파일의 모든 엘리먼트의 계층적인 목록이다. 각 엘리먼트 트리의 형식은 다음과 같다.

    a. **index number** (예 (11))는, 태그 레퍼런스로 각 태그마다 인덱스를 붙여 놓은 것이다. 태그 레퍼런스는 이 번호순서대로 설명한다.

    b. XML SCHEMA 에서 정의한 XML 태그 명을 <element name>형식으로 표시한다.

    c. XML SCHEMA 에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element.

    d. 몇몇 태그에는 “P”문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝 할 때 사용된다.

2. **element 레퍼런스** 는 각 XML element 마다 하나의 테이블로 구성되어 있다. 이 테이블은 다음과 같은 세부 항목들을 포함하고 있다.

    a. **Description**: XML 태그에 대한 간단한 설명

    b. **Value Description**: XML 태그에 입력하는 값과 타입

    c. **Value Type**: 값의 데이터 타입

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위한 추천 값

h. **Child Elements**: 자신의 태그 안에 다시 태그를 포함하는 경우에 사용 가능한 태그

3. **sample XML file**: 구체적인 "subjects.xml"의 예제.

D.2 XML SCHEMA/XML Tree

```
(1) <subjects>
     (2) <subject>*
           (3) <description>?
           (4) <principal>+
                 (5) <classname>?
                 (6) <name>
           (7) <credential>*
                 (8) <classname>?
                 (9) <property>*
                       (10) <name>
                       (11) <value>?
```

D.3 Element Reference

(1) <subjects>

Description	Subject(users)들을 정의한다.
Child Elements	(2)subject*

(2) <subjects> <subject>

Description	Subject 를 정의한다.

Child Elements	(3)description? (4)principal+ (7)credential*

(3) <subjects> <subject> <description>

Description	Subject 에 대한 설명.
Value Description	임의의 설명문
Value Type	token
Example	<description>A really nide guy!</description>

(4) <subjects> <subject> <principal>

Description	해당 Subject 에 대한 user identify 를 정의한다.처음으로 등장하는 principal 태그가 "primary" identify 이다.
Child Elements	(5)classname? (6)name

(5) <subjects> <subject> <principal> <classname>

Description	java.security.Principal 인터페이스를 구현한 Java 클래스명. 이 클래스는 String 타입의 Principal 명을 받아 들이는 public 생성자를 반드시 가지고 있어야 한다.
Value Description	Java 클래스명
Value Type	token
Example	<classname>mypackage.MyPrincipalImpl</classname>

(6) <subjects> <subject> <principal> <name>

Description	해당 Principal 명 (가령, 사용자명 또는 그룹명)
Value Description	A String id.
Value Type	token
Example	<name>johan</name>

(7) <subjects> <subject> <credential>

Description Subject 에 대한 Credential 을 정의한다.

Child Elements
```
(8)classname?
(9)property*
```

(8) <subjects> <subject> <credential> <classname>

Description jeus.security.base.CredentialFactory 인터페이스를 구현한 Java 클래스명. 어떤 파라미터도 갖지 않는 public 생성자를 제공해야 한다.

Value Description Java 클래스명

Value Type token

Default Value jeus.security.resource.PasswordFactory

Example
```
<classname>mypackage.MyPasswordFactory</classname>
```

(9) <subjects> <subject> <credential> <property>

Description 선택사항으로 Credential Factory 에 넘겨줄 name-value 쌍의 속성값을 정의한다. 속성에 대한 자세한 정보는 Credential Factory 관련 클래스의 문서를 참고하기 바란다.

Child Elements
```
(10)name
(11)value?
```

(10) <subjects> <subject> <credential> <property> <name>

Description 속성명

Value Description 속성명

Value Type token

Example
```
<name>password</name>
```

(11) <subjects> <subject> <credential> <property> <value>

Description 선택사항으로 속성에 대한 구체적인 값

Value Description 속성 . "password" 속성 값은 Base 64 로 인코딩 되어야 한다.

Value Type token

Example
```
<value>bXlwYXNzd29yZA==</value>
```

D.4 Sample subjects.xml File

<<subjects.xml>>

```
<subjects>
    <subject>
        <description>A really nice guy!</description>
        <principal>
            <classname>mypackage.MyPrincipalImpl</classname>
            <name>johan</name>
        </principal>
        <credential>
            <classname>mypackage.MyPasswordFactory</classname>
            <property>
                <name>password</name>
                <value>bXlwYXNzd29yZA==</value>
            </property>
        </credential>
    </subject>
</subjects>
```

E policies.xml

E.1 소개

본 레퍼런스 부록은 보안 시스템에서 Policy 권한체크와 관련된 "policies.xml" 의 모든 XML 태그를 설명한다. 해당 "policies.xsd" 파일은 JEUS_HOME\config\xsds 디렉토리에 있다.

본 부록은 다음의 3 가지 하위 절로 구성되어 있다.

1. **XML SCHEMA/XML tree 는**, XML 구성 파일의 모든 엘리먼트의 계층적인 목록이다. 각 엘리먼트 트리의 형식은 다음과 같다.

    a. **index number** (예 (11))는, 태그 레퍼런스로 각 태그마다 인덱스를 붙여 놓은 것이다. 태그 레퍼런스는 이 번호순서대로 설명한다.

    b. XML SCHEMA 에서 정의한 XML 태그 명을 <element name>형식으로 표시한다.

    c. XML SCHEMA 에서 정의한 Cardinality 를 표시한다. “?” = 0 개나 1 개의 element, “+” = 1 개 이상의 element, “*” = 0 개 이상의 element, (기호가 없음) = 정확히 1 개의 element.

    d. 몇몇 태그에는 “P”문자를 붙여 놓았는데, 해당 태그는 성능에 관계되는 태그라는 것을 뜻한다. 이 태그는 설정을 튜닝 할 때 사용된다.

2. **element 레퍼런스** 는 각 XML element 마다 하나의 테이블로 구성되어 있다. 이 테이블은 다음과 같은 세부 항목들을 포함하고 있다.

    a. **Description**: XML 태그에 대한 간단한 설명

    b. **Value Description**: XML 태그에 입력하는 값과 타입

    c. **Value Type**: 값의 데이터 타입

d. **Default Value**: 해당 XML 을 사용하지 않았을 때 기본적으로 사용되는 값

e. **Defined values**: 이미 정해져 있는 값

f. **Example**: 해당 태그에 대한 예

g. **Performance Recommendation**: 성능 향상을 위한 추천 값

h. **Child Elements**: 자신의 태그 안에 다시 태그를 포함하는 경우에 사용 가능한 태그

3. **sample XML file**: 구체적인 "policies.xml" 의 예제.

E.2 XSD/XML SCHEMA/XML Tree

```
(1) <policies>
      (2) <policy>*
            (3) <role-permissions>?
                  (4) <role-permission>*
                        (5) <principal>+
                        (6) <role>
                        (7) <actions>?
                        (8) <classname>?
                        (9) <excluded>?
                        (10) <unchecked>?
            (11) <resource-permissions>*
                  (12) <context-id>?
                  (13) <resource-permission>*
                        (14) <role>*
                        (15) <resource>
                        (16) <actions>?
                        (17) <classname>?
                        (18) <excluded>?
                        (19) <unchecked>?
```

E.3 Element Reference

(1) <policies>

Description JEUS 의 권한 체크와 관련된 Policy 들을 정의한다.

Child Elements (2)policy*

(2) <policies> <policy>

Description JEUS 의 Policy 를 정의한다.

Child Elements (3)role-permissions?
(11)resource-permissions*

(3) <policies> <policy> <role-permissions>

Description 해당 Policy 에 대한 principal-role 매핑들을 정의한다.

Child Elements (4)role-permission*

(4) <policies> <policy> <role-permissions> <role-permission>

Description 해당 Policy 에 대한 principal-role 매핑을 정의한다.

Child Elements (5)principal+
(6)role
(7)actions?
(8)classname?
(9)excluded?
(10)unchecked?

(5) <policies> <policy> <role-permissions> <role-permission> <principal>

Description role 에 해당하는 user principal 을 지정한다.

Value Description security 의 subjects.xml 에서 지정되어 있는 principal 이름

Value Type token

(6) <policies> <policy> <role-permissions> <role-permission> <role>

Description principal 들에게 부여할 role 이름을 지정한다.

Value Type token

(7) <policies> <policy> <role-permissions> <role-permission> **<actions>**

Description 이 role permission 객체에 대한 action 을 정의한다. default 로 사용되는 role permission 은 정해진 action 이 없다.

Value Type token

(8) <policies> <policy> <role-permissions> <role-permission> **<classname>**

Description 사용할 role permission class name 을 지정한다. 지정하지 않으면 JEUS 에서 기본적으로 제공하는 class 가 사용된다.

Value Type token

(9) <policies> <policy> <role-permissions> <role-permission> **<excluded>**

Description role 을 사용하지 못하도록 만든다.

Example
```
<excluded/>
```

(10) <policies> <policy> <role-permissions> <role-permission> **<unchecked>**

Description 아무런 체크없이 role 을 사용가능하도록 만든다.

Example
```
<unchecked/>
```

(11) <policies> <policy> **<resource-permissions>**

Description 해당 Policy 에 대한 role-resource 매핑들을 정의한다.

Child Elements
```
(12)context-id?
(13)resource-permission*
```

(12) <policies> <policy> <resource-permissions> **<context-id>**

Description role-resource 매핑이 적용 되는 context id(보통 이 태그는 사용되지 않는다).

Value Description context id 값

Value Type token

Default Value default

Example
```
<context-id>MyContext</context-id>
```

(13) <policies> <policy> <resource-permissions> **<resource-permission>**

Description 해당 Policy 에 대한 role-resource 매핑을 정의한다.

Child Elements
```
(14)role*
(15)resource
(16)actions?
(17)classname?
(18)excluded?
(19)unchecked?
```

(14) <policies> <policy> <resource-permissions> <resource-permission> <role>

Description 해당 Resource 에 매핑되는 Role

Value Description Role 명

Value Type token

Example
```
<role>Administrator</role>
```

(15) <policies> <policy> <resource-permissions> <resource-permission> <resource>

Description Role 을 맵핑하는 Resource.

Value Description Resource 명.

Value Type token

Example
```
<resource>jeus.server.*</resource>
```

(16) <policies> <policy> <resource-permissions> <resource-permission> <actions>

Description 선택적인 값으로 resource permission 클래스의 생성자에 넘길 actions 값

Value Description resource permission class 에 넘길 actions 데이터 값.

Value Type token

Example
```
<actions>boot,down</actions>
```

(17) <policies> <policy> <resource-permissions> <resource-permission> <classname>

Description java.security.Permission 을 상속한 Java 클래스명. 이 클래스는 resource permission 에 사용된다.

Value Description Java 클래스명

Value Type	token
Default Value	jeus.security.resource.ResourcePermission
Example	`<classname>jeus.security.resource.TimeConstrainedResourcePermission</classname>`

(18) <policies> <policy> <resource-permissions> <resource-permission> **<excluded>**

Description	만약 이 태그가 있으면, 해당 Resource 는 배제된다(누구도 해당 Resource 에 접근할 수 없다).
Value Description	어떤 값도 설정되지 않는다. empty 타입.

(19) <policies> <policy> <resource-permissions> <resource-permission> **<unchecked>**

Description	만약 이 태그가 있으면, 해당 Resource 는 체크되지 않는다(누구나 이 Resource 에 접근할 수 있다).
Value Description	어떤 값도 설정되지 않는다. empty 타입.

E.4 Sample policies.xml File

<<policies.xml>>

```
<policies>
    <policy>
        <role-permissions>
            <role-permission>
                <principal>peter</principal>
                <role>Administrator</role>
                <actions>09:00-17:00</actions>
                <classname>
              jeus.security.resource.TimeConstrainedRolePermission
                </classname>
                <excluded/>
                <unchecked/>
            </role-permission>
        </role-permissions>
        <resource-permissions>
```

```
            <context-id>MyContext</context-id>
            <resource-permission>
                <role>Administrator</role>
                <resource>jeus.server.*</resource>
                <actions>boot,down</actions>
                <classname>
          jeus.security.resource.TimeConstrainedResourcePermission
                </classname>
                <excluded/>
                <unchecked/>
            </resource-permission>
        </resource-permissions>
    </policy>
</policies>
```

F 기본 SPI 구현 레퍼런스

F.1 소개

본 부록은 jeus.security.impl 패키지에 포함되어 있는 디폴트 보안 서비스 구현 클래스들을 설명한다. 이러한 서비스는 master.xml 과 slave.xml 에서 설정된다.

본 부록의 구조는 다음과 같다.

- "F.2"와 같은 각 절은 SPI 명을 타이틀로 하고 있다. 해당 절에서는 SPI 를 구현하고 있는 디폴트 서비스 클래스들에 대한 자세한 정보를 제공한다. 가령, "F.2 jeus.security.spi.SecurityInstaller" 절은 jeus.security.spi.SecurityInstaller 를 구현하거나 상속하고 있는 모든 클래스에 대해 설명한다.
- "F.2.1"과 같은 하위 절은 SPI 를 구현하고 있는 구체적인 클래스를 소개한다. 하위 절의 제목은 패키지명을 포함한 전체 클래스명이며, 실제 보안 시스템에서 서비스로 master.xml 과 slave.xml 에 등록된다.
- SPI 구현 클래스에 대해 소개하고 있는 각 하위 절은 다음과 같은 세 부항목을 가지고 있다.
  - **Description**: 구현 클래스에 대한 설명과 구현 클래스가 제공하려는 서비스에 대한 소개.
  - **Used in Location**: "client", "master", "slave" 중의 하나이거나, 두군데 이상이 올 수 있다. "client"는 해당 서비스가 client JVM 에서, "master"는 master JVM 에서, "slave"는 slave JVM 에서 각각 실행된다는 의미이다. 각 용어에 대한 설명은 이 문서의 도입부 '용어 설명란과 3.4 절을 참고하기 바란다.
  - **Dependencies**: 구현 클래스가 호출하는 다른 SPI 클래스의 목록. 가령, "I"라는 구현 클래스가 SPI "S"를 호출한다면, 보안 서비스를 제공하기 위해서 SPI "S"도 함께 설치해야 한다.

- **System Properties**: 구현 클래스가 사용하는 시스템 속성들. 시스템 속성은 JEUS_HOME\bin 디렉토리내의 각 종 JEUS 관련 스크립트에서 설정될 수 있다.

- **Service Properties**: 구현 클래스에서 사용되는 속성들의 목록. 속성명, 유효한 속성값, 속성에 대한 설명을 포함한다. 속성명과 값은 PropertyHolder 인터페이스를 구현하는 Service 클래스의 서브 클래스를 통해 설정되고 읽혀진다. 속성은 master.xml 과 slave.xml 에서 설정되며, 속성명은 `<security-service>` 태그 내의 `<property><name>` 태그로, 속성값은 `<property><value>`태그로 설정된다(서비스 속성을 설정하는 더 자세한 정보는 3.4 절을 참고하기 바란다).

F.2 jeus.security.spi.SecurityInstaller

F.2.1 jeus.security.impl.installer.DefaultSecurityInstaller

Description: 이 클래스는 SecurityInstaller SPI 의 디폴트 구현 클래스이다. 각 보안 도메인에서 JEUS_HOME\config\<node_name>\ security\<domain_name>아래에 있는 master.xml, slave.xml 에 등록된다.

Used in Location: master, slave, client JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.3 jeus.security.spi.LoginService

F.3.1 jeus.security.impl.login.ClientLoginService

Description: 이 클래스는 LoginService SPI 의 client 측 구현 클래스로써, 특정 Subject 로 로그인하자 마자 사용자 인증을 일어나게 한다 (NoAuthClientLoginService 와는 반대이다).

Used in Location: Client JVM

Dependencies: None.

System Properties: None

Service Properties: None

F.3.2 jeus.security.impl.login.NoAuthClientLoginService

Description: 이 클래스는 LoginService SPI 의 client 측 구현 클래스로써, 로그인시 client 측에서 어떤 사용자 인증 과정도 일어나지 않게 한다. 따라서 어떤 Subject 도 사용자 인증 과정 없이 강제로 로그인 된다. 실제 사용자 인증 과정은 Subject 를 Request 와 함께 서버측으로 전파 하면서 서버측에서 일어나게 된다.

Used in Location: Client JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.3.3 jeus.security.impl.login.ServerLoginService

Description: 이 클래스는 LoginService SPI 의 서버측 구현 클래스이다.

Used in Location: Master and slave JVM.

Dependencies: AuthenticationService, AuthorizationService, SubjectValidationService.

System Properties: None

Service Properties: None

F.4 jeus.security.spi.SubjectValidationService

F.4.1 jeus.security.impl.validation.ClientSubjectValidationService

Description: client 측 SubjectValidationService 로써, 모든 원격지 서버로의 호출을 위임한다.

Used in Location: Client JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.4.2 jeus.security.impl.lockout.SubjectLockoutValidationService

Description: SubjectValidationService 의 일종으로, Subject 가 jeus.security.resource.Lock Credential 을 가지고 있는 경우, 예외를 발생시킨다. SubjectLockoutService 는 Subject 에 자동으로 락을 추가하기 위해 사용한다(자세한 것은 아래를 보도록 한다).

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.4.3 jeus.security.impl.expiration.SubjectExpirationValidationService

Description: SubjectValidationService 의 일종으로 Subject 가 유효기간이 만료된 jeus.security.resource.ExpiryTime Credential 을 가지고 있을 경우, 예외를 발생시킨다.

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.5 jeus.security.spi.SubjectFactoryService

F.5.1 jeus.security.impl.subfactory.PasswordSubjectFactoryService

Description: SubjectFactoryService 란 JAAS CallbackHandler 에게 사용자 명과 패스워드를 요청해서, 새로운 Subject 를 생성하고, 해당 Subject 에 jeus.security.resource.Password 를 생성해서 할당한다. JAAS CallbackHandler 를 설정하는 방법은 jeus.security.base.Environment 클래스를 참고하기 바란다.

Used in Location: Client JVM.

Dependencies: SecurityInstaller.

System Properties: None

Service Properties: None

F.6 jeus.security.spi.AuthenticationService

F.6.1 jeus.security.impl.atn.ClientAuthenticationService

Description: 원격지 서버로 사용자 인증을 위임하는 client 측 사용자 인증 서비스이다.

Used in Location: Client JVM.

Dependencies: LoginService.

System Properties: None

Service Properties: None

F.6.2 jeus.security.impl.atn.DefaultAuthenticationService

Description: 디폴트 서버측 사용자 인증 메커니즘이다.

Used in Location: Master and slave JVM.

Dependencies: LoginService, CredentialMappingService, AuthenticationRepositoryService, CredentialVerificationService.

System Properties: None

Service Properties: None

F.7 jeus.security.spi. AuthenticationRepositoryService

F.7.1 jeus.security.impl.atnrep.ClientAuthenticationRepositoryService

Description: 모든 AuthenticationRepositoryService 메소드 호출을 원격지 서버에 위임한다.

Used in Location: Client JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.7.2 jeus.security.impl.atnrep.MemoryAuthenticationRepositoryService

Description: 런타임시 로컬 메모리에 Subject 들을 저장한다.

Used in Location: Master and slave JVM.

Dependencies: LoginService.

System Properties: None

Service Properties: None

F.7.3 jeus.security.impl.atnrep. DistributedMemoryAuthenticationRepositoryService

Description: MemoryAuthenticationRepositoryService 를 확장한 것으로, 클러스터로 묶여 있는 모든 JVM 에 Repositoy 변경 사항을 일괄 통보 하는 기능이 추가되어 있다.

Used in Location: Master and slave JVM.

Dependencies: LoginService

System Properties: None

Service Properties: “synchtrigger”: 원격지 서버에서 서비스가 시작하는 시간과 로컬 서버에서 서비스가 시작하는 시간 사이에, 허용되는 최소한의 시간 차이로 milliseconds 로 나타낸다. 해당 서비스는 원격지 서버로부터 Subject 를 다운로드 받기 때문에 어느정도 시간 차이가 존재한다. 디폴트 값은 "15000" (15 초)이다. 이 속성은 start-up 퍼포먼스를 향상시키는데 이용된다.

F.7.4 jeus.security.impl.atnrep. XMLPersistedDistributedMemoryAuthenticationRepositoryService

Description: DistributedMemoryAuthenticationRepositoryService 를 확장한 것으로 Subject 데이터를 XML 파일로 읽고 쓸수 있는 기능이 추가되었다(기본 XML 파일은 subjects.xml 이다).

Used in Location: Master and slave JVM.

Dependencies: SecurityInstaller.

System Properties: None

Service Properties: “filename”: Subject 데이터를 저장하는 파일명으로 절대경로로 지정한다. 만약 설정되어 있지 않으면, 디폴트 값으로 해당 도메인의 "subjects.xml"이 지정된다.

F.8 jeus.security.spi.CredentialMappingService

F.8.1 jeus.security.impl.credmap.JKSCertificateCredentialMappingService

Description: 인증서 저장소로부터 인증서에 대한 alias 를 패치하여, Subject 에 java.security.cert.Certificate Credential 을 할당한다.

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: “truststore”: 인증서를 저장하고 있는 JKS 키저장소의 파일명(절대 경로로 표시). 이 속성이 지정되지 않으면, 디폴트로 현재 보안 도메인이 설정되어 있는 디렉토리가 사용된다.

F.9 jeus.security.spi. CredentialVerificationService

F.9.1 jeus.security.impl.verification.PasswordVerificationService

Description: 두개의 jeus.security.resource.Password 인스턴스가 동일한지 체크한다.

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.10 jeus.security.spi.AuthorizationService

F.10.1 jeus.security.impl.azn.ClientAuthorizationService

Description: 원격지 서버로의 모든 AuthorizationService 호출을 위임한다.

Used in Location: Client JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.10.2 jeus.security.impl.azn.DefaultAuthorizationService

Description: 기본 서버측 사용자 인증 메커니즘.

Used in Location: Master and slave JVM.

Dependencies: LoginService, AuthorizationRepositoryService.

System Properties: None

Service Properties: None

F.10.3 jeus.security.impl.azn.JACCAuthorizationService

Description: JACC Policy 결정 및 집행 규약을 지원한다. JACC Provider를 사용할 경우에만 설정된다.

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.11 jeus.security.spi. AuthorizationRepositoryService

F.11.1 jeus.security.impl.aznrep.ClientAuthorizationRepositoryService

Description: 원격지 서버로의 모든 AuthorizationRepositoryService 호출을 위임한다.

Used in Location: Client JVM.

Dependencies: None.

System Properties: None

Service Properties: None

F.11.2 jeus.security.impl.aznrep.MemoryAuthorizationRepositoryService

Description: 로컬 런타임 메모리에 Policy 목록을 저장한다.

Used in Location: Master and slave JVM.

Dependencies: LoginService.

System Properties: None

Service Properties: None

F.11.3 jeus.security.impl.aznrep. DistributedMemoryAuthorizationRepositoryService

Description: MemoryAuthorizationRepositoryService 를 확장한 것으로, 클러스터로 묶인 모든 JVM 에 저장소 변경 사항을 일괄 통보하는 기능이 추가되어 있다.

Used in Location: Master and slave JVM.

Dependencies: LoginService.

System Properties: None

Service Properties: “synchtrigger”: 원격지 서버에서 서비스가 시작하는 시간과 로컬 서버에서 서비스가 시작하는 시간 사이에, 허용되는 최소한의 시간 차이로 milliseconds 로 나타낸다. 해당 서비스는 원격지 서버

로부터 Policy 를 다운로드 받기 때문에 어느정도 시간 차이가 존재한다. 디폴트 값은 "15000" (15 초)이다. 이 속성은 start-up 퍼포먼스를 향상시키는데 이용된다.

F.11.4 jeus.security.impl.aznrep.XMLPersistedDistributedMemoryAuthorizationRepositoryService

Description: DistributedMemoryAuthorizationRepositoryService 를 확장한 것으로 XML 파일로 Policy 데이터를 읽고 쓰는 기능이 추가되었다 (디폴트 XML 파일은 policies.xml 이다).

Used in Location: Master and slave JVM.

Dependencies: SecurityInstaller.

System Properties: None

Service Properties: “filename”: Policy 데이터를 저장하는 파일의 절대경로로써, 디폴트값은 해당 도메인 설정 디렉토리에 있는 "policies.xml" 이다.

F.11.5 jeus.security.impl.aznrep.JACCAuthorizationRepositoryService

Description: JACC Policy 설정 계약을 지원한다. JACC Provider 를 사용하는 경우에만 설정된다.

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: “filename”: 디폴트 시스템에서 Policy 파일의 절대경로. 디폴트값은 해당 도메인 설정 디렉토내의 "policies.xml"이다.

F.12 jeus.security.spi.EventHandlingService

F.12.1 jeus.security.impl.auditlog.BasicAuditLogFileService

Description: 현재 도메인 디렉토리내의 “audit.log” 파일에 보안관련 이벤트를 덤프뜨는 것이다.

Used in Location: Master and slave JVM.

Dependencies: None.

System Properties: None

Service Properties: “level”: 이벤트 로그가 남겨지는 최소한의 심각도 레벨. 레벨은 “fatal”, “serious”, “warning”, “information” , “debug”. 중의 하나로 설정한다. 디폴트 값은 "warning"으로 "warning"이상의 심각도인 "fatal", "serious", "warning" 에 해당하는 이벤트 로그를 남긴다. "information", "debug" 레벨은 "warning"보다 레벨이 낮으므로 로그로 남지 않는다.

F.12.2 jeus.security.impl.lockout.SubjectLockoutService

Description: 각 사용자가 사용자 인증에 실패한 횟수를 세어서, 그 값이 허용하는 최대 횟수를 초과하면, jeus.security.resource.LockFactory 가 Subject 에 추가되어, 해당 사용자로 더 이상의 로그인 시도를 허락하지 않는다. 이때, 해당 도메인에는 SubjectLockoutValidationService 가 함께 설정되어 있어야 한다.

Used in Location: Master and slave JVM.

Dependencies: LoginService, AuthenticationRepositoryService.

System Properties: None

Service Properties: “maxattempts”: 사용자 인증시 허용하는 최대 실패 횟수로써, 디폴트 값은 "3"이다.

G JEUS Server Permissions 레퍼런스

G.1 소개

본 부록은 표준 Permission 리소스명과 리소스 액션을 소개한다. 이들은 JEUS 서버 모듈(JNDI, JMS, manager, security 등.)이 리소스에 대한 접근 권한을 가지고 있는지 체크하는 데 이용된다.

리소스 Permission 체크와 관련된 Permission 타입은 “jeus.security.resource.ResourcePermission”이며, context id 는 항상 "default"이다.

G.2 Checked Security System Permission

디폴트 보안 시스템을 사용할 때, 서버측 보안 시스템에서 체크하는 Permission 목록이다.

표 2 Checked Security System Permissions

Resource name	Resource action	Check time
`jeus.security.spi.AuthenticationRepositoryService`	`getSubject`	저장소로부터 Subject 에 대한 정보를 가져 오려고 할 때
`jeus.security.spi.AuthenticationRepositoryService`	`addSubject`	저장소에 새로운 Subject 데이터를 추가하려할 때
`jeus.security.spi.AuthenticationRepositoryService`	`removeSubject`	저장소로부터 Subject 데이터를 삭제하려할 때

Resource name	Resource action	Check time
jeus.security.spi.AuthorizationRepositoryService	getPolicy	저장소로부터 Policy 에 대한 정보를 가져 오려고 할 때
jeus.security.spi.AuthorizationRepositoryService	addPolicy	저장소에 Policy 데이터를 추가하려 할 때
jeus.security.spi.AuthorizationRepositoryService	removePolicy	저장소로부터 Policy 데이터를 삭제하려 할 때

G.3 Checked JEUS Manager Permissions

Resource name	Resource action	Check time
jeus.manager.JeusServer	boot	JEUS Server 를 부팅하려 할 때
jeus.manager.JeusServer	down	JEUS Server 를 다운하려 할 때
jeus.manager.JeusServer	backup	JEUS Server 에 대한 백업 노드를 생성하고자 할 때
jeus.manager.JeusServer	join	JEUS 클러스터에 새로운 JEUS 서버를 추가하려 할 때 ("dynamic booting")
jeus.manager.JeusServer	die	JEUS Server 를 일시 정지 하고자 할 때

Resource name	Resource action	Check time
jeus.manager.JeusServer	exit	JEUS manager 를 종료하고자 할 때 ("jeusexit")
jeus.manager.JeusServer	resurrect	JEUS Server 가 부팅에 실패해서, 재부팅하려할 때

G.4 Checked JNDI Server Permissions

Resource name	Resource action	Check time
jeus.ejb.JNDIServer	start	JNDI 엔진/서버를 부팅하고자 할 때
jeus.ejb.JNDIServer	down	JNDI 엔진/서버를 다운하고자 할 때

G.5 Checked EJB Server Permissions

Resource name	Resource action	Check time
jeus.ejb.EJBServer	start	EJB 엔진/서버를 부팅하고자 할 때
jeus.ejb.EJBServer	down	EJB 엔진/서버를 다운하고자 할 때

G.6 Checked JMS Server Permissions

Resource name	Resource action	Check time
jeus.jms.client.createJMSCon.	createJMSCon	JMS 서버에 커넥션을 맺고자 할 때

H 보안 이벤트 레퍼런스

H.1 소개

본 부록은 SPI 클래스와 디폴트 보안 서비스 구현 클래스로부터 EventHandlingService 로 방출되는 표준 보안 이벤트(Security Event)에 대해 소개한다. 이 내용은 "jeus.security.spi.EventHandlingService" SPI 를 구현하여 자신만의 이벤트 핸들링을 개발하는데 유용하게 참고할 수 있다.

목록의 양식은 다음과 같다.

G.2.X <Event type> = 이벤트 타입

Source Class: 이벤트가 발생한 클래스

Event Type: 이벤트 타입

Event Level: 이벤트 레벨 (FATAL, SERIOUS, WARNING, INFORMATION, DEBUG).

Event Context: 이벤트 context 에 대한 key-value 쌍의 집합.

Emitted When? 이벤트가 발생하는 조건

jeus.security.base.Event 클래스와 jeus.security.spi.EventHandlingService 클래스에 대한 더 자세한 정보는 Javadoc 을 참고하기 바란다.

Note: 보통 이벤트는 이벤트 소스와 같은 도메인에 있는 EventHandlingService 들로 방출된다. 그러나, 예외로 security.install.successful 과 security.uninstall.attempt 이벤트는 보안 시스템에 설정되어 있는 모든 도메인으로 방출된다.

H.2 이벤트

H.2.1 security.validation.failed

Source Class: jeus.security.spi.SubjectValidation Service

Event Type: security.validation.failed

Event Level: WARNING

Event Context: *Key*: "subject" *Value*: 유효성 검증이 실패한 jeus.security.base.Subject

Emitted When? SubjectValidationService 가 SecurityException.을 발생할 때 마다.

H.2.2 security.authentication.failed

Source Class: jeus.security.spi.AuthenticationService

Event Type: security.authentication.failed

Event Level: WARNING

Event Context: *Key*: "subject" *Value*: 사용자 인증이 실패한 jeus.security.base.Subject

Emitted When? Subject 에 대한 사용자 인증이 실패할 때마다.

H.2.3 security.authorization.failed

Source Class: jeus.security.spi.AuthorizationService

Event Type: security.authentication.failed

Event Level: WARNING

Event Context:
Key: "contextid" *Value*: Permission 체크가 일어난 context id.

Key: "permission" *Value*: 체크될 java.security.Permission;

Key: "subject" *Value*: 사용자 인증이 실패한 jeus.security.base.Subject

Emitted When? 사용자 인증이 실패할 때마다.

H.2.4 security.authentication.repository.subject.added

Source Class: jeus.security.spi.AuthenticationRepositoryService

Event Type: security.authentication.repository.subject.added

Event Level: INFORMATION

Event Context: *Key*: "subject" *Value*: 추가된 jeus.security.base.Subject

Emitted When? AuthenticationRepositoryService 에 Subject 가 성공적으로 추가될 때 마다.

H.2.5 security.authentication.repository.subject.removed

Source Class: jeus.security.spi.AuthenticationRepositoryService

Event Type: security.authentication.repository.subject.removed

Event Level: INFORMATION

Event Context: *Key*: "subject" *Value*: 삭제된 jeus.security.base.Subject .

Emitted When? AuthenticationRepositoryService 로부터 Subject 가 성공적으로 삭제될 때마다.

H.2.6 security.authentication.repository.subject.removed.complete

Source Class: jeus.security.spi.AuthenticationRepositoryService

Event Type: security.authentication.repository.subject.removed.complete

Event Level: INFORMATION

Event Context: *Key*: "name" *Value*: 성공적으로 삭제된 Subject.

Emitted When? Subject 가 AuthenticationRepositoryService 로부터 성공적으로 삭제될 때마다.

H.2.7 security.authorization.repository.policy.added

Source Class: jeus.security.spi.AuthorizationRepositoryService

Event Type: security.authorization.repository.policy.added

Event Level: INFORMATION

Event Context: *Key*: "policy" *Value*: 추가된 jeus.security.base.Policy.

Emitted When? AuthorizationRepositoryService 에 새로운 Policy 가 추가될 때 마다.

H.2.8 security.authorization.repository.policy.removed

Source Class: jeus.security.spi.AuthorizationRepositoryService

Event Type: security.authorization.repository.policy.removed

Event Level: INFORMATION

Event Context: *Key*: "policy" *Value*: 삭제된 jeus.security.base.Policy.

Emitted When? AuthorizationRepositoryService 로부터 Policy 데이터가 삭제될 때 마다.

H.2.9 security.authorization.repository.policy.removed.complete

Source Class: jeus.security.spi.AuthorizationRepositoryService

Event Type: security.authorization.repository.policy.removed.complete

Event Level: INFORMATION

Event Context: *Key*: "contextid" *Value*: 저장소로부터 삭제된 java.lang.String 타입의 context id

Emitted When? AuthorizationRepositoryService 로부터 context id 가 삭제될 때마다.

H.2.10 security.install.successful

Source Class: jeus.security.spi.SecurityInstaller

Event Type: security.install.successful

Event Level: INFORMATION

Event Context: None.

Emitted When? 보안 시스템이 성공적으로 설치된 후.

H.2.11 security.uninstall.attempt

Source Class: jeus.security.spi.SecurityInstaller

Event Type: security.uninstall.attempt

Event Level: INFORMATION

Event Context: None.

Emitted When? 보안 시스템이 제거되기 전.

I 보안 시스템 속성 레퍼런스

I.1 소개

본 부록은 JEUS_HOME\bin 디렉토리에 있는 JEUS 시작 스크립트에서 -D 옵션으로 설정할 수 있는 모든 표준 Java 및 JEUS 보안 관련 속성들에 대해 설명한다.

I.2 Standard J2SE & J2EE Security Properties

Property	Description	Example
`java.security.manager`	코드 보호를 통해 JEUS 서버의 견고성을 향상시키기 위해서 J2SE SecurityManager 를 사용한다. 이는 퍼포먼스를 떨어뜨릴 수 있다. 디폴트로, SecurityManager 는 JEUS 에서 사용되지 않는다.	`java.security.manager`
`java.security.policy`	SecurityManager 에서 사용하는 J2SE policy 파일의경로를 설정한다. 디폴트는 JAVA_HOME\lib\security\java.security 이다.	`java.security.policy=c:\MyPolicy`
`javax.security.jacc.policy.provider`	JACC Policy 구현 클래스명을 설정한다.	`javax.security.jacc.policy.provider=myprovider.MyJACCPolicy`

Property	Description	Example
javax.security.jacc.PolicyConfigurationFactory.provider	JACC PolicyConfigurationFactory 구현 클래스명을 설정한다.	javax.security.jacc.PolicyConfigurationFactory.provider=myprovider.MyJACCPolicyConfigurationFactory

I.3 Standard JEUS Security System Properties

Property	Description	Example
jeus.security.default.configDir	보안 시스템이 사용하는 루트 디렉토리를 설정한다. 디폴트는 JEUS_HOME\config\<nodename>\security 디렉토리이다.	jeus.security.default.configDir=c:\MyDecurityDir
jeus.security.default.domainName	디폴트 도메인명을 설정한다. 디폴트는 “SYSTEM_DOMAIN”이다.	jeus.security.default.domainName=MY_DOMAIN

Property	Description	Example
jeus.security.globalPassword	클러스터로 묶여 있는 JEUS Server 의 전역 시스템 패스워드를 설정한다. 이 패스워드는 같은 클러스터 내에 있는 모든 JEUS Server 에서 동일해야 한다. JEUS client container 에는 적용되지 않는다. 이 설정의 디폴트값은 "globalpass"이다. 이 속성값은 "CodeSubject" 에 로그인하기 위해 securityadmin 스크립트에 임시적으로 설정되어 있다. 전역 패스워드는 반드시 기밀성을 유지해야 하며, 디폴트이외의 값으로 다시 설정해야 한다.	jeus.security.globalPassword=mysystempassword123
jeus.security.installer.classname	SecurityInstaller 의 구현 클래스명을 설정한다.	jeus.security.installer.classname=mypackage.MySecurityInstaller
jeus.security.jacc.principalRoleMapper	.jeus.security.impl.aznrep.JACCPrincipalRoleMapper 인터페이스를 구현하는 클래스명을 설정한다. 이 클래스는 JACC Provider 에서 principal-role 매핑을 담당한다.	jeus.security.jacc.principalRoleMapper=myprovider.MyJACCPrincipalToRoleMapper

Note: 속성명과 속성값은 JEUS_HOME\bin 에 있는 JEUS 실행 스크립트(Window 의 경우 jeus.cmd, Unix 의 경우 jeus)에 입력해야 하며, 반드시 -D 로 시작해야 한다. 속성들은 반드시 한 라인에 입력되어야 한다.

J Security API&SPI Reference

JEUS Security Javadoc API 문서는 매우 방대하기 때문에, 여기서 다루지 않는다. 웹 브라우저로 JEUS 가 설치된 디렉토리의 docs/securitydoc/index.html 파일을 열어 API 문서를 참고하기 바란다.

색 인