

Security 안내서

JEUS 9

TMAXSOFT

저작권 공지

Copyright 2024. TmaxSoft Co., Ltd. All Rights Reserved.

제한된 권리

이 소프트웨어(JEUS®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

Java, Solaris는 Oracle Corporation 및 그 자회사, 관계회사의 등록 상표입니다.

Microsoft, Windows, Windows NT는 Microsoft Corporation의 등록 상표 또는 상표입니다.

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상호, 상표, 또는 등록 상표입니다.

본 사용설명서에 기재된 회사, 시스템, 제품 이름 등에 반드시 상표 표시(™, ®)를 하지는 않습니다.

오픈 소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다.: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :

`${INSTALL_PATH}/license/oss_licenses`

기술 지원

홈페이지: <https://www.tmaxsoft.com>

기술 서비스 센터: 1544-8629

안내서 이력

제품 버전	안내서 버전	발행일	비고
JEUS 9	3.1.1	2024-09-27	-

목차

1. 보안 시스템 소개	1
1.1. 개요	1
1.2. 주요 특징	1
1.3. 시스템 구조	2
1.4. 주요 개념	4
1.4.1. 로그인	4
1.4.2. 인증	5
1.4.3. 권한 체크(부여)	6
1.4.4. 보안 감사	12
1.4.5. 서비스와 SPI	12
1.4.6. 도메인	14
1.5. 성능 향상과 수준 향상	15
1.5.1. 성능 향상	15
1.5.2. 수준 향상	16
2. 보안 시스템 설정	18
2.1. 개요	18
2.2. 보안 도메인 설정	19
2.2.1. XML 편집 설정	19
2.2.2. 사용자 계정 및 보안 정책 설정	21
2.3. 보안 도메인 구성요소 설정	22
2.3.1. XML 편집 설정	22
2.4. 보안 서비스 설정	24
2.4.1. XML 편집 설정	24
2.5. 보안 시스템 사용자 정보 설정	28
2.5.1. XML 편집 설정	28
2.5.2. 데이터베이스 사용 설정	30
2.5.3. 비밀번호 보안 설정	31
2.5.4. 로그인 정보 캐시 기능	36
2.6. 보안 시스템 정책 설정	36
2.6.1. XML 편집 설정	37
2.6.2. 데이터베이스 사용 설정	42
2.7. 추가 항목 설정	43
2.7.1. Java SE SecurityManager 설정	43
2.7.2. JACC Provider 설정	44
2.7.3. Identity 부여를 위한 정보 설정	44
2.7.4. Identity에 대한 인증서 정보 설정	45
3. 애플리케이션과 모듈에서 보안 설정	48
3.1. 개요	48
3.1.1. 모듈 Deployment 대 애플리케이션 Deployment	48

3.1.2. Role-to-Resource 매핑	48
3.1.3. Principal-to-Role 매핑	50
3.1.4. 사용자 설정	52
3.2. EJB 모듈 보안 설정	53
3.2.1. ejb-jar.xml 설정	53
3.2.2. jeus-ejb-dd.xml 설정	56
3.3. 웹 모듈 보안 설정	58
3.3.1. web.xml 설정	59
3.3.2. jeus-web-dd.xml 설정	62
3.4. Jakarta EE 애플리케이션 보안 설정	64
3.4.1. application.xml 설정	65
3.4.2. jeus-application-dd.xml 설정	66
3.5. 예제	67
4. 보안 시스템 API 프로그래밍	69
4.1. 개요	69
4.2. Java SE Permission 설정	69
4.3. 기본 API	69
4.4. 리소스 API	70
4.5. SPI 클래스	71
4.6. 예제	71
5. Custom 보안 서비스 개발	74
5.1. 개요	74
5.2. 서비스 클래스	74
5.3. Custom 보안 서비스 구현 패턴	76
5.4. SPI 클래스	77
5.4.1. SubjectValidationService SPI	77
5.4.2. SubjectFactoryService SPI	78
5.4.3. AuthenticationService SPI	78
5.4.4. AuthenticationRepositoryService SPI	79
5.4.5. IdentityAssertionService SPI	79
5.4.6. CredentialMappingService SPI	80
5.4.7. CredentialVerificationService SPI	80
5.4.8. AuthorizationService SPI	81
5.4.9. AuthorizationRepositoryService SPI	81
5.4.10. EventHandlingService SPI	81
5.4.11. Dependencies between SPI Implementations	82
5.5. 보안 서비스 설정	83
6. JACC Provider 사용	84
6.1. 개요	84
6.2. JACC 규약	84
6.2.1. Provider 설정 규약	84

6.2.2. Policy 설정 규약	85
6.2.3. Policy 결정 및 집행 규약	85
6.3. JACC Provider 개발	86
6.3.1. JACC Provider 구현	86
6.3.2. JACC Provider 패키징	88
6.3.3. Default JACC Provider	88
6.4. JEUS 보안 시스템과 JACC Provider 통합	88
7. JAAS 사용	92
7.1. 개요	92
7.2. JEUS-LDAP 연동 위한 LoginModule 구현	93
7.3. LDAP JAAS LoginModule 서비스 설정	97
7.4. 데이터베이스 연동 위한 LoginModule 구현	99
7.5. 데이터베이스 LoginModule 서비스 설정	104
Appendix A: 보안 이벤트 서비스	107
A.1. 개요	107
A.2. 이벤트	107
Appendix B: JEUS Server Permissions	111
B.1. 개요	111
B.2. JEUS 시스템 리소스 이름	111
B.3. jeusadmin 명령어 권한 설정	112
참고 자료	114

1. 보안 시스템 소개

본 장에서는 보안 시스템에 대한 소개와 주요 특징, 구조에 대해 설명한다.

1.1. 개요

일상 생활에서 보안이라는 개념은 중요한 무엇인가가 외부 또는 내부로부터의 침입이나 훼손 등의 우려가 있을 때에 이를 차단하고 예방하기 위해서 사용된다. 이러한 보안의 개념은 컴퓨터를 사용하거나 엔터프라이즈 환경에서 사용자들의 정보를 다루고 서비스를 제공하는 데에도 유사하게 적용될 수 있다.

이때 보안의 대상이 되는 것은 사용자들의 이름과 비밀번호, 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.

JEUS의 보안 서비스는 기본적으로 SPI(Service Provider Interface)에 근거하여 서비스되고 있으며, JEUS에 등록된 리소스와 사용자의 정보를 보호하기 위해서 다양한 보안 서비스를 제공한다.

본 안내서는 JEUS 보안 구조와 보안 정보 관리에 대한 일반적인 정보에 대해 자세히 설명하고 있다.

JEUS 보안 시스템은 다음의 목표를 두고 설계되었다.

- 유연하고 교체하기 쉬운 프레임워크여야 한다.

기존의 3rd-party 보안 메커니즘을 장착한 보안 시스템과 효율적으로 통합되고, 다양한 데이터 스토리지를 지원해야 한다.

- 기밀성이 유지되어야 한다.

뛰어난 해커가 좋지 않은 의도를 가지고 보안 시스템 소스를 디컴파일해도 권한이 없는 시스템에는 결코 접근할 수 없어야 한다.

- 성능에 무리가 없어야 한다.

시스템과 애플리케이션 코드에 보안 시스템이 관여해도 전체 성능에는 최소한의 영향만 미쳐야 한다. 일반적으로 보안이 강화될수록 성능이 떨어지는 경향이 있다. 최대한의 보안 기능을 제공하면서 가능한 한 빨리 처리할 수 있어야 한다.

- 유지 보수를 쉽게 하고 3rd-party 보안 서비스를 쉽게 생성하기 위해 간결하고 명료한 API와 SPI가 제공되어야 한다.
- 데이터의 내용이 보존되고 무결성이 보장되어야 한다.
- 표준을 준수해야 한다.

JEUS 보안 시스템은 위에서 언급한 설계상의 목적을 상당히 충족하고 있다.

1.2. 주요 특징

JEUS 보안 시스템의 주요 특징은 다음과 같다.

- 공개된 아키텍처와 유연한 프레임워크를 가지고 있어서 기존에 사용 중인 3rd-party 보안 시스템과 효과적으로 통합할 수 있다.
- 동적인 Principal-to-Role, Role-to-Resource 매핑을 지원한다.

동적인 매핑은 실제 권한 부여 매핑(Principal-to-Role, Role-to-Resource 매핑)이 런타임에 적용된다는 것을 의미한다.

이는 다음과 같은 보안 정책을 가능하게 한다.

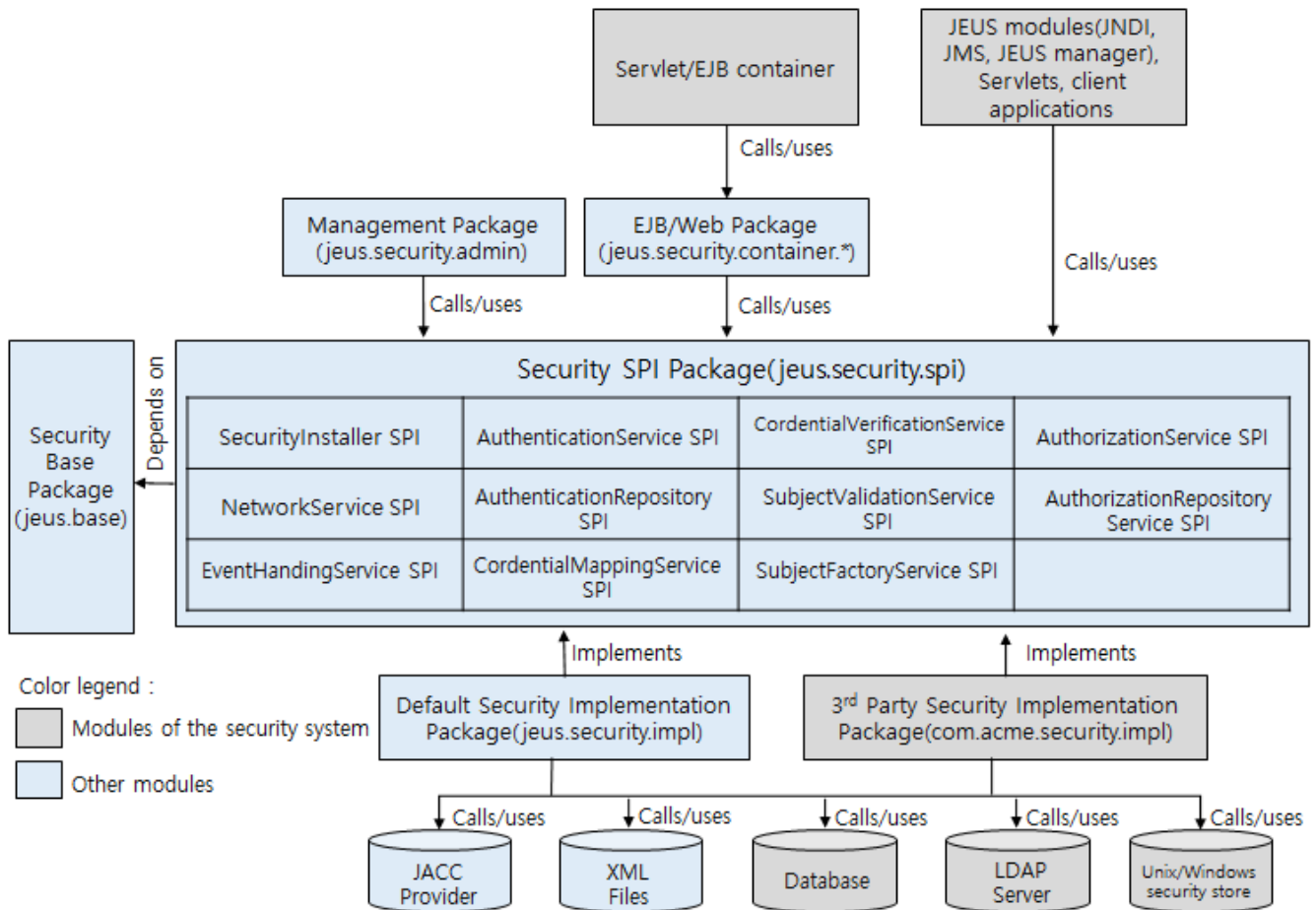
- 사용자 U는 월요일부터 금요일까지 업무시간인 9시부터 5시까지 R이란 Role을 부여받는다.
- 모든 사람이 R이란 Role을 부여받는다.
- 아무도 R이란 Role을 부여받을 수 없다.
- 보안 도메인(이하 도메인) 개념을 지원한다. 이는 서로 다른 Jakarta EE 애플리케이션들이 각각의 특성에 맞는 별도의 보안 서비스를 이용할 수 있게 한다.

JEUS 시스템에서의 도메인과 보안 도메인은 서로 다른 개념이므로 혼동하지 않도록 주의한다. 시스템에서의 도메인은 서버 관리 단위를 의미하고, 보안 도메인은 보안 관리 단위를 의미한다.

- 인증, 권한 부여, Repositories, 감사, 클러스터링 등과 같은 중요한 보안 기능에 대한 디폴트 구현을 제공한다.
- 유연한 이벤트 핸들링 모델을 통한 보안 감사 메커니즘을 지원한다.
- 서블릿, EJB, 애플리케이션 소스 코드 내에 직접 보안 기능을 추가할 수 있다. 이는 Subject와 Policy를 추가하는 것과 같은 단순한 보안 기능에 한해 가능하다.
- Java SE 8과 JACC 2.0 스펙을 완벽히 지원한다.
- 본 안내서와 추가적인 JavaDoc을 통해 문서화를 충분히 지원하고 있다.
- 다른 JEUS 모듈과 독립적이다. 따라서, JEUS 시스템에서 보안 시스템만 따로 분리하여 다른 context에 적용하기가 상대적으로 쉽다.

1.3. 시스템 구조

다음은 보안 시스템의 기본 아키텍처이다.



보안 시스템 구조

위의 그림에서 주요 컴포넌트(Package)는 다음과 같다.

- jeus.security.spi

보안 시스템의 가장 핵심 부분인 보안 SPI 클래스로 중앙에 큰 박스로 구분되어 있다. 이 추상 클래스들에는 3rd-party 제품에서 구현해야 하는 추상 메소드뿐 아니라 JEUS 컨테이너를 포함한 사용자 코드에서 호출하는 메소드들도 포함되어 있다.

현재 11개의 사용 가능한 SPI 클래스가 있으며, 각각은 인증, 권한 부여와 같은 보안의 핵심적인 기능을 담당하고 있다.

- jeus.security.base

왼쪽의 작은 박스로 구분되어 있으며 보안 SPI가 참조하는 인터페이스와 구현 클래스가 포함되어 있다. 이 패키지에서 중요한 2가지 클래스는 Subject와 Policy이다.

- jeus.security.impl.*

아래쪽 가운데 박스로 구분되어 있으며 JEUS에서 제공하는 SPI 클래스의 디폴트 구현 클래스가 포함되어 있다. 디폴트 구현 클래스는 현재 XML 파일 Repository와 JACC Provider를 지원하고 있다.

- jeus.security.admin, jeus.security.container

위쪽 가운데 있는 박스로 구분되어 있으며 SPI 클래스를 호출하는 코드를 포함하고 있다.

- Repository와 외부 보안 메커니즘

그림의 가장 하단에 있는 것은 Repository와 외부 보안 메커니즘을 나타내고 있다.

Repository로는 데이터베이스, LDAP 서버, XML 파일이 있다. Repository는 Subject와 Policy와 같은 보안 요소들을 영구적으로 저장할 때 사용된다.

외부 보안 메커니즘은 인증 또는 권한 부여가 실행되는 메커니즘을 의미한다. 외부 보안 메커니즘의 대표적인 예는 JACC Provider이다. 일반적으로 Repository와 보안 메커니즘 사이의 경계를 명확히 나누기는 힘들다. SPI 구현 클래스(여기서는 jeus.security.impl.* 패키지에 포함된 클래스)가 실제 어떤 Repository와 보안 메커니즘을 적용할지 결정한다.

1.4. 주요 개념

본 절에서는 보안 시스템을 이해하는 데 필수적인 보안 시스템 아키텍처와 관련된 중요한 개념에 대해 설명한다.

- 로그인
- 인증
- 권한 체크(부여)
- 보안 감사
- 서비스와 SPI
- 도메인

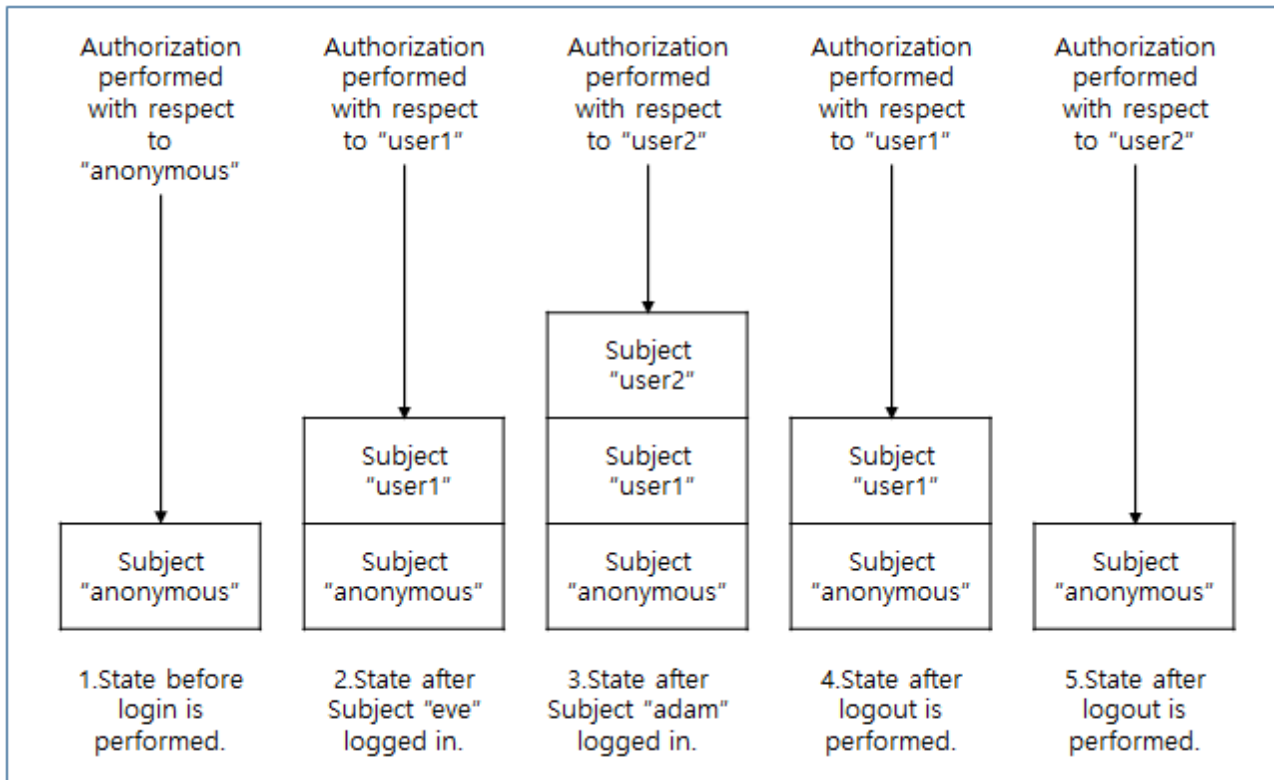
1.4.1. 로그인

JEUS 보안 시스템에서 로그인은 Subject를 실행 Thread(Java Thread)에 결합시키는 것을 의미한다. 이 개념은 인증과는 다르다. 일반적으로 Subject가 실행 Thread와 결합되기 전에 인증이 일어난다. 만약 성공적으로 인증되면 로그인은 정상적으로 진행되어 Subject와 Thread가 결합하게 되지만, 인증에 실패하면 로그인은 더 이상 진행되지 않는다.

Subject가 성공적으로 로그인한 후에 로그인된 Subject를 대상으로 권한 체크가 이루어진다. 따라서, 로그인은 인증과 권한 체크, 양쪽 모두에 걸쳐있다고 할 수 있다. 로그인의 반대 개념은 로그아웃으로 현재 Subject를 실행 Thread로부터 분리하는 과정이다.

로그인 이면에는 다음과 같은 Stack 기반 오퍼레이션이 있다. 여러 개의 서로 다른 Subject로 로그인했을 경우 최근에 로그인한 Subject부터 Stack의 최상단에 쌓인다. 그 후 권한 체크에는 Stack의 최상단에 놓인 Subject(가장 최근에 로그인한 Subject)를 사용한다. 이 Subject가 로그아웃하고 나면, Stack으로부터 제거되고 바로 직전에 로그인한 Subject가 Export되어 활성화된다.

다음은 이 메커니즘에 대한 설명이다.



Stack 기반의 로그인 메커니즘

JEUS 보안 시스템에서 Java Thread마다 하나의 로그인 Stack을 가질 수 있다. 이전의 보안 아키텍처와 유사하게 로그인 메커니즘은 jeus.security.impl.login.CommonLoginService에 의해 구현된다.

1.4.2. 인증

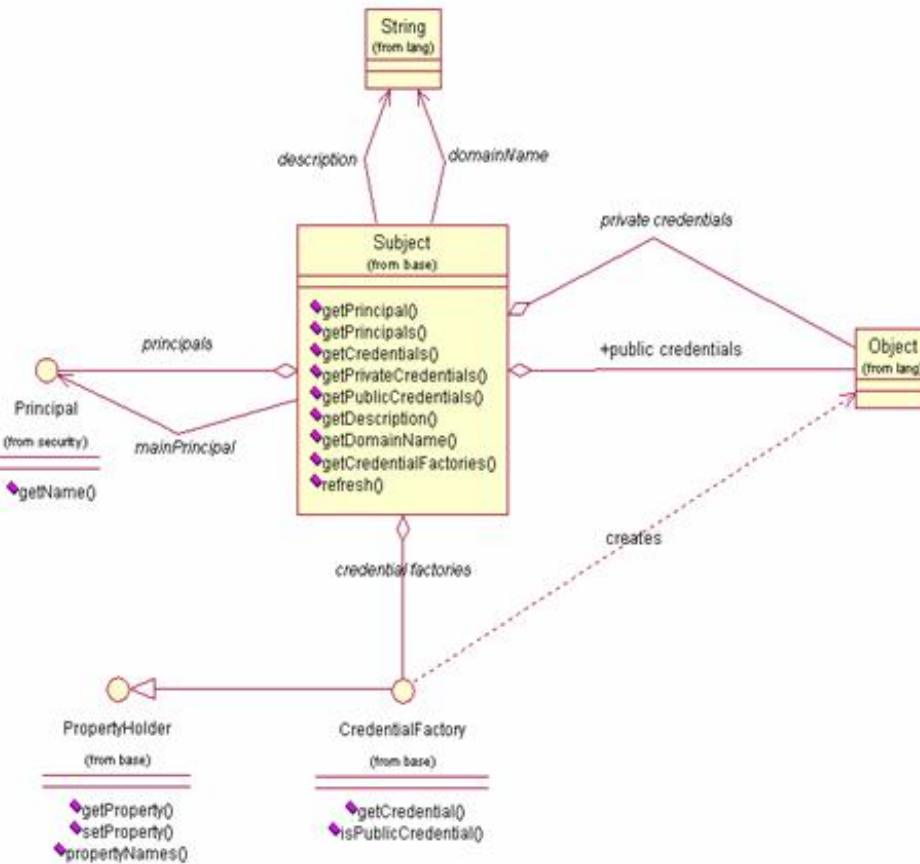
인증은 호출자의 신원을 파악하는 작업으로 이는 이후 권한을 체크할 때 사용된다.

JEUS 보안 시스템에서 신원은 java.security.Principal 인터페이스로 정의한 Principal을 의미한다. Subject는 이러한 Principal들을 속성값으로 저장하고 있다. Subject는 jeus.security.base.Subject 클래스를 의미하며, 로그인할 때 실행 Thread와 결합하는 주체이기도 하다. Subject는 항상 Principals과 Credentials을 보안 속성으로 포함하고 있다.



JEUS 보안 시스템에서의 Subject는 javax.security.auth.Subject 클래스가 정의하는 JAAS의 Subject와는 다른 것이다. 그러나 이 두 클래스는 많은 공통점이 있어서 한 쪽에서 다른 쪽으로 전환하는 것이 가능하고, 일부 정보를 잃더라도 Alias로 지칭될 수 있다.

JEUS Subject는 UML로 다음과 같이 표기할 수 있다.



Subject UML 다이어그램

Subject는 유일한 ID인 메인 Principal을 가지고 있다. 또한 메인 Principal 외의 여러 개의 부가적인 Principal을 가질 수 있는데, 이러한 부가적인 Principal들은 다른 Subject와 공유할 수 있다(따라서, 부가적인 Principal을 그룹 Principal이라고도 한다).

Subject는 또한 public 또는 private credentials을 가질 수 있다. Credential은 보통 Subject의 신원을 확인하거나, 특정 정보를 전달하려는 목적으로 사용되는 증빙 자료이다. 대표적인 private credential은 패스워드가 있고, public credential은 디지털 인증서가 있다.

Subject는 소위 CredentialFactory를 사용하여 Credential들을 생성한다. Subject의 refresh()라는 메소드가 실제 CredentialFactory로부터 Credential을 얻어 와서, 이를 public 또는 private credential set에 추가한다.



Subject는 반드시 하나의 도메인에 속해야 한다는 것에 주의한다. 도메인 A에 속하는 Principal "user1"은 도메인 B에 속하는 Principal "user1"과는 별개의 것이다.

1.4.3. 권한 체크(부여)

JEUS 보안 시스템에서 권한 체크는 이미 성공적으로 인증된 Subject가 특정 액션을 실행시킬 권한이 있는지 여부를 확인하는 것이다.

권한 체크는 보통 시스템 레벨에서 일어난다. 특정 Subject(보통 administrators)가 JEUS 서버를 기동 또는

종료시킬 권한을 가지고 있는지 여부를 체크하는 것이다. 또한 권한 체크는 애플리케이션 레벨에서도 일어난다. 원격지 호출자가 특정 애플리케이션 컴포넌트(특정 EJB 메소드의 실행, 특정 Servlet 호출)에 접근할 수 있는지 여부를 JEUS 엔진에서 확인하는 것은 여기에 해당한다.

Jakarta EE에서와 마찬가지로 JEUS 보안 시스템에서 권한 부여는 Role-Based 메커니즘이다. 즉, Jakarta EE 애플리케이션 Assembler 또는 개발자가 Role에 따라 보안 설정(security restriction)을 하면 시스템 관리자가 애플리케이션을 deploy하면서 실제 시스템의 Principal을 논리적인 Role에 매핑한다.

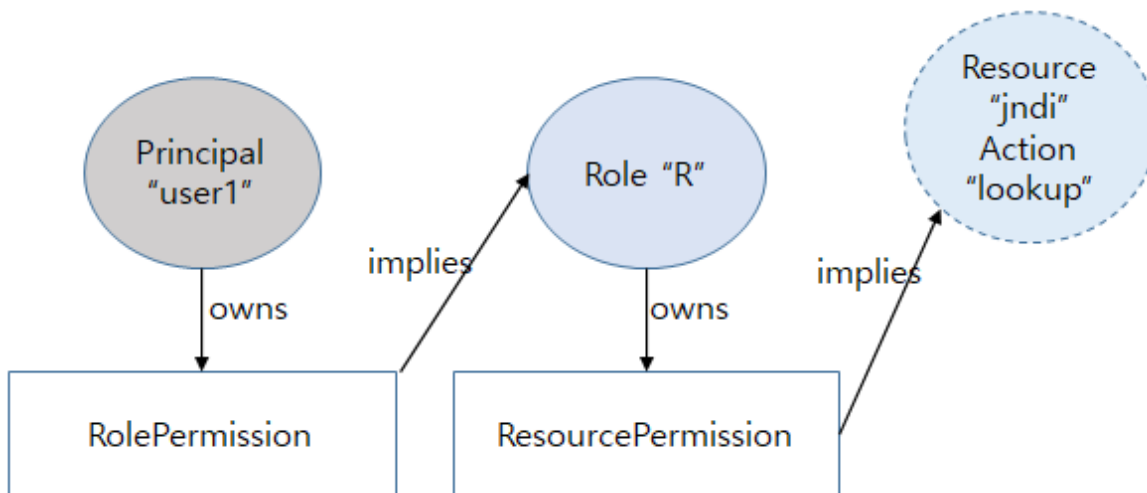


Role-Based 접근방법은 Jakarta EE 애플리케이션에서 뿐만 아니라 JEUS 시스템과 관련된 권한 체크에도 사용된다는 것에 주의한다.

더 나아가 JEUS 보안 시스템에서 개개의 권한 매핑을 Permission(실제 java.security.Permission의 서브 클래스)이라 한다. 예를 들어 Principal "user1"은 "R"이라 불리는 Role에 접근할 수 있는 RolePermission을 가지고 있다고 가정한다(이를 "user1"은 Role "R"에 포함되어 있다고 한다).

그리고 Role "R"은 Resource "jndi" 리소스에 접근해 Action "lookup"을 실행할 수 있는 Resource Permission을 가지고 있다고 가정한다.

다음은 이러한 개념에 대한 설명이다.



Role 기반 Permission 체크

위 그림에서 보듯이 Principal과 Role만이 실제 물리적 실체로서 모델링된다. 그림에서 리소스를 감싸고 있는 원은 점선으로 표시되어 있는데, 리소스는 실제 권한 부여 시스템에서 물리적으로 모델링되는 부분이 아닌 암시되는 부분이기 때문이다.

“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, Principal은 여러 개의 RolePermission을 가지고 있고 각 RolePermission은 특정 Role이 Principal을 포함하는 것을 허용한다. 다시 Role도 여러 개의 ResourcePermission을 가지고 있는데, 각 ResourcePermission은 해당 Role이 특정 리소스들에 대해 특정 액션들을 취할 수 있도록 허락한다. 따라서 Principal-Role-Resource를 연결해서 생각해 보면, 어떤 Principal이 특정 리소스에 대한 특정 액션을 실행할 수 있을지 여부를 판단할 수 있다.

위의 그림에서 Principal "user1"은 실선으로 RolePermission을 소유하고 있다(owns)라고 표시되어 있다. 그러나 RolePermission은 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 RolePermission이 해당 Role을 암시할지 여부는 동적으로 결정된다는 의미이다.

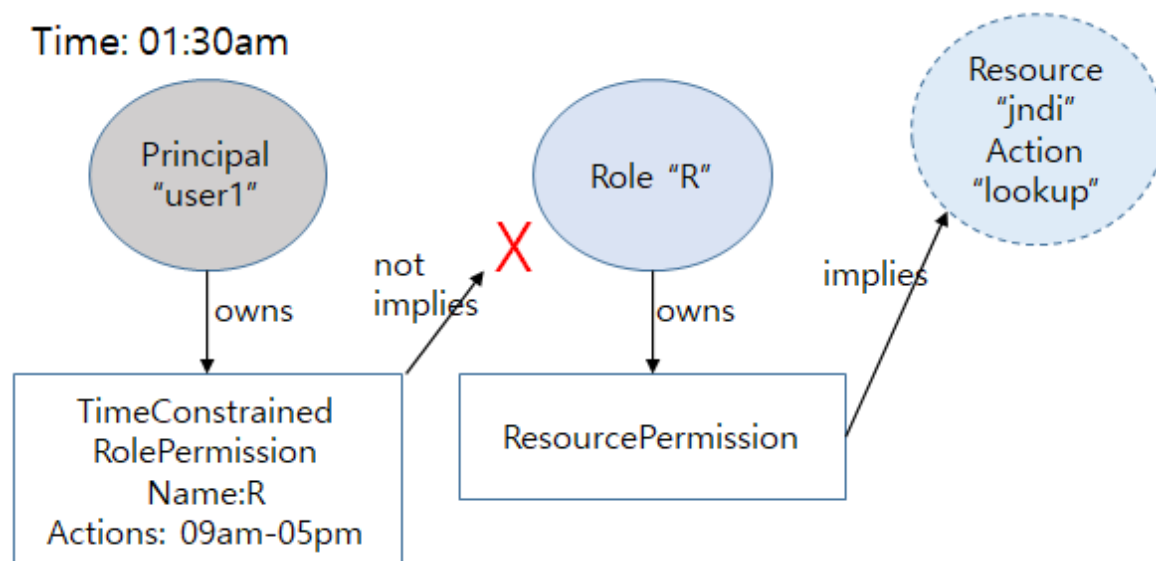
실제로 RolePermission의 `implies(Permission p)` 메소드에 의해 결정되는데 `implies()` 메소드가 `true`를 리턴하면 RolePermission이 Role "R"을 암시하고, `false`를 리턴하면 암시하지 않는다. 따라서 전자의 경우에는 Principal "user1"이 Role "R"에 포함되지만, 후자의 경우에는 포함되지 않는다. 같은 논리가 Role "R"과 Resource "jndi" 사이에도 성립된다.



Permission에 대한 자세한 내용은 Java SE JavaDoc에서 `java.security.Permission` 부분을 참고한다.

위의 정보를 바탕으로 동적인 매핑을 구현하기란 실제 어려운 일이 아니다. 단지 `implies()` 메소드의 구현을 변경하면, 특정 조건하에서만 Permission이 Role 또는 Resource를 암시하도록 만들 수 있다. 예를 들어 동적인 매핑을 이용해서 Principal "user1"이 업무 시간(9시부터 5시) 동안만 Role "R"에 포함되게 구현할 수 있다. 이는 새로운 RolePermission 서브 클래스(이하 `TimeConstrainedRolePermission`)를 생성하고, `implies()` 메소드 내에서 시간 제약과 관련된 코드만 작성하면 된다.

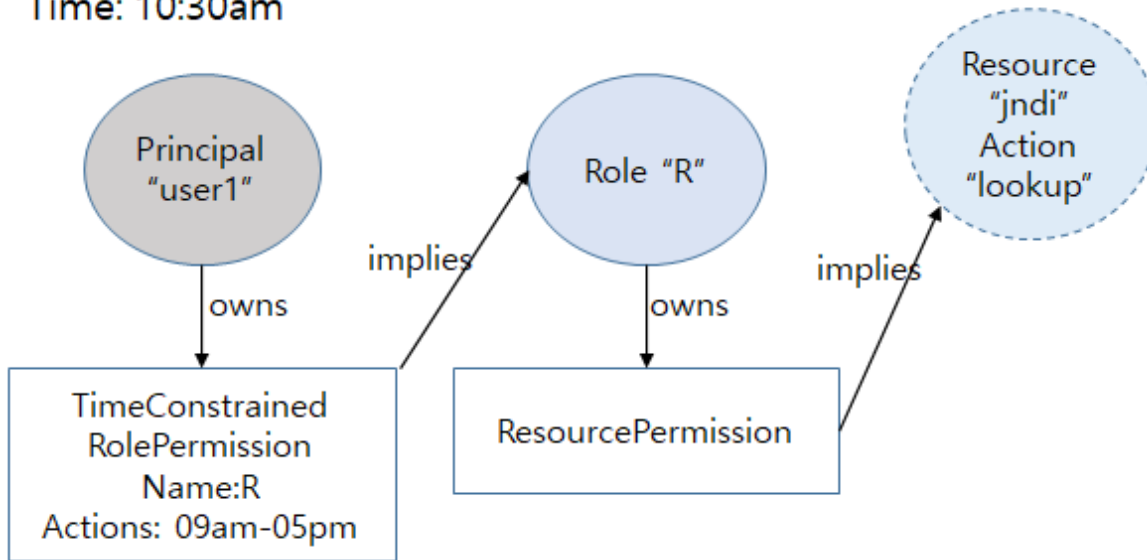
예로 **오전 1시 30분의 Role**와 **오전 10시 30분의 Role**을 보기 바란다.



오전 1시 30분의 Role

오전 10시 30분의 Role에서 보듯이 오전 1시 30분에는 Principal "user1"은 Role "R"의 권한이 없다.

Time: 10:30am



오전 10시 30분의 Role

TimeConstrainedRolePermission은 두 가지 값인 Name과 Actions를 가지고 있다. Name은 Target Role의 이름인 "R"이고, Actions는 유효한 시간(9시부터 5시까지)을 나타내고 있다. Name과 Actions는 대부분의 java.security.Permission 구현 클래스의 변수들로 선언된다.

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

구분	설명
Excluded Permission	<p>누구도 해당 Permission에 대한 허가를 가질 수 없다.</p> <p>예로 Resource "RSC"에 대한 접근을 Excluded Permission으로 설정했다면, 아무도 Resource "RSC"에 접근할 수 없게 된다.</p> <p>Excluded Permission은 Unchecked Permission보다 우선순위가 높다.</p>
Unchecked Permission	<p>누구나 해당 Permission에 대한 허가를 가질 수 있다.</p> <p>예로 Resource "RSC"에 대해 Unchecked Permission이 설정되어 있다면, Role에 상관없이 누구나 Resource "RSC"에 접근할 수 있게 된다.</p> <p>Unchecked Permission은 Excluded Permission보다 낮은 우선순위를 가지나, Checked Permission보다 높은 우선순위를 가진다.</p>
Checked Permission	<p>특정 Principal이나 Role에만 허가된 Permission으로 지금까지 계속 설명한 Permission이 여기에 해당한다.</p>

이에 대한 몇 가지 예제들은 다음에 더 살펴본다.

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

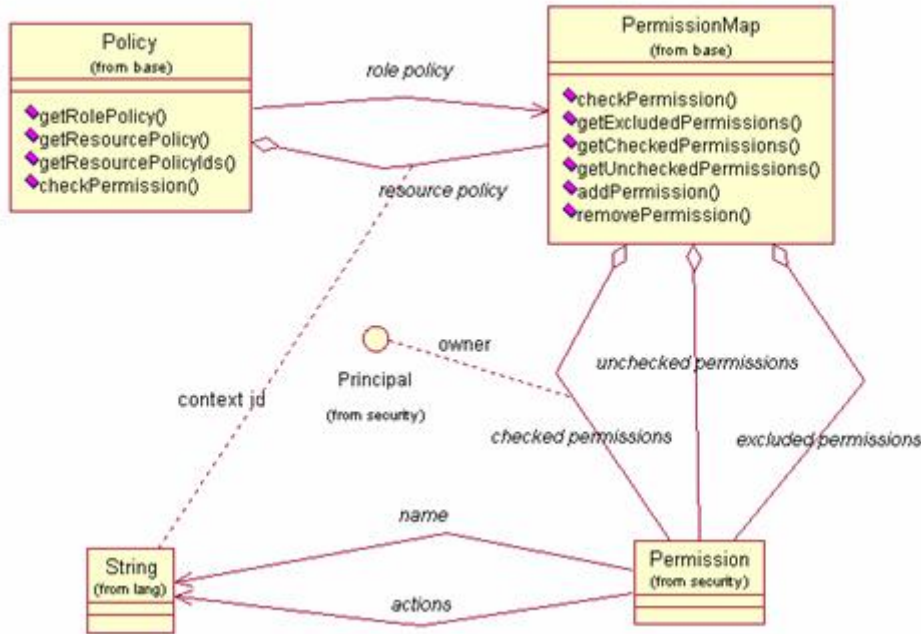
Policy는 2가지 종류의 PermissionMap을 가지고 있다.

- Principal-to-Role Map(Role Policy0)

- Role-to-Resource Map(Resource Policy)

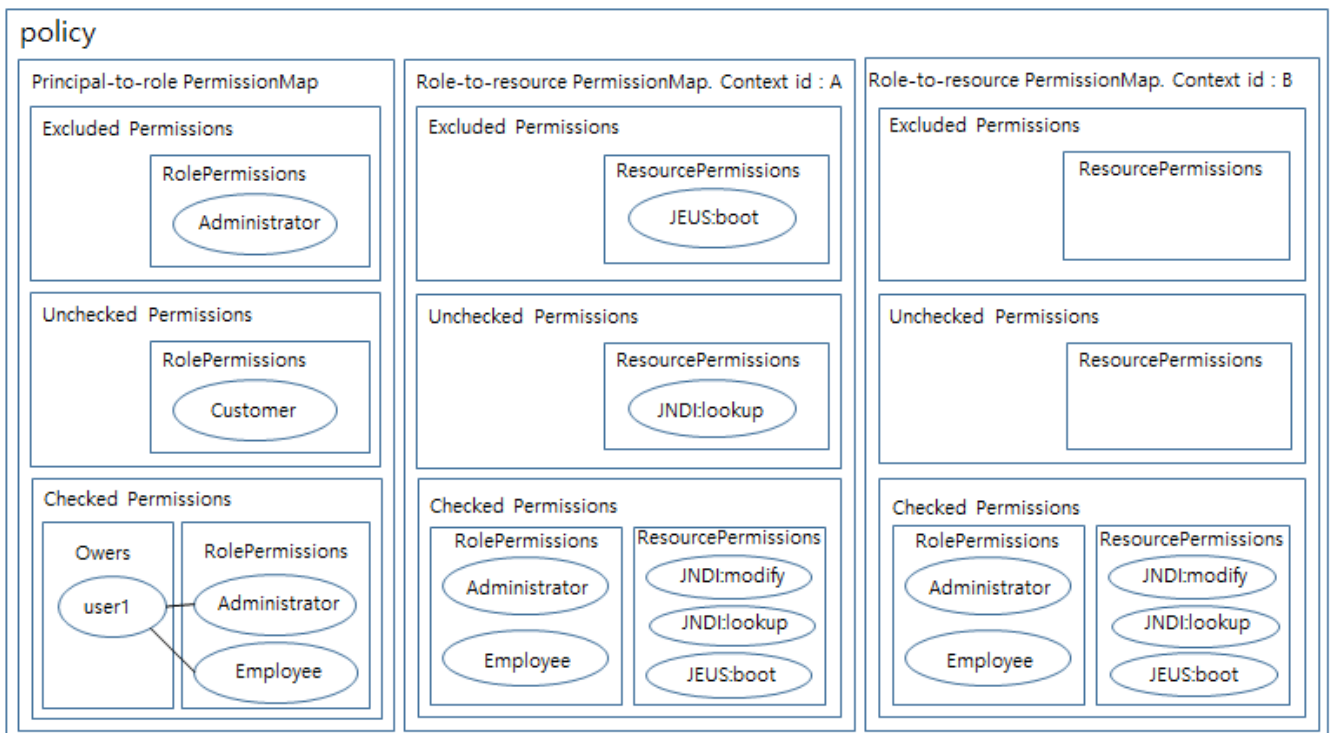
이들은 각각 Principal-to-Role 매핑과 Role-to-Resource 매핑을 나타낸다. 각 PermissionMap은 다시 위에서 언급한 3가지 종류(Excluded, Unchecked, Checked)의 Permission들을 포함하고 있다. Checked Permission의 경우 Permission과 RolePermissionMap은 Principal이나 Role을 매개로 결합되어 있고, Policy와 ResourcePermissionMap은 Context ID(권한 체크가 일어나는 범위를 나타낸다)를 통해 결합되어 있다.

다음은 Policy와 PermissionMap의 UML 다이어그램을 나타낸다.



Policy와 PermissionMap의 UML 다이어그램

이상에서 언급한 내용이 다음 그림에 요약되어 있다.



하나의 Principal-to-Role Map과 2개의 Role-to-Resource를 가진 Policy 예제

해당 Policy는 하나의 Principal-to-Role Map(필수 사항)과 각각의 Context ID가 "A"와 "B"인 2개의 Role-to-Resource Map을 포함하고 있다. 3가지 PermissionMap은 다음과 같은 Permission set을 가지고 있다.

- Excluded Permissions
- Unchecked Permissions
- Checked Permissions

박스 내의 타원은 Name과 Action을 가진 실제 Permission 인스턴스(Name: Action)를 나타내고 있다.

Context ID가 "A"이고, Principal이 "user1"인 Subject가 "JNDI"에 접근해서 "modify"라는 액션을 실행하고 싶다고 가정했을 때 과연 "user1"은 원하는 액션을 실행할 수 있을까?

권한 부여 시스템은 이 요구사항을 다음과 같은 방식으로 풀어나간다.

1. 이름이 "JNDI"이고 액션이 "modify"인 새로운 ResourcePermission(RSP)을 생성한다.
2. RSP는 Context ID "A"와 Principal "user1"과 함께 Policy에 넘겨진다.
3. Context ID "A"에 대한 Role-to-ResourcePermissionMap이 Policy에서 선택된다.
4. Policy는 PermissionMap "A"의 Excluded Permission에 RSP가 포함되는지 체크한다. 체크 결과 RSP는 Excluded Permission이 아니다.
5. Policy는 PermissionMap "A"의 Unchecked Permission에 RSP가 포함되는지 체크한다. 체크 결과 RSP는 Unchecked Permission이 아니다.
6. Policy는 PermissionMap "A"의 Checked Permission에 RSP가 포함되는지 체크한다. 체크 결과 RSP는 Checked Permission 중에 하나이다.
7. Policy는 RSP를 암시하는 Permission의 소유자를 알아낸다. "Administrator"라 불리는 Role이 바로 소유자이다.
8. Policy는 "Administrator"라는 이름을 가진 RolePermission(RLP)을 생성한다.
9. Policy는 Principal-to-RolePermissionMap을 꺼내온다.
10. Policy는 Principal-to-RolePermission의 Excluded Permission에 포함되는지 체크한다. 체크 결과 RLP는 Excluded Permission이다.
11. Policy는 "Administrator"라는 Role이 Excluded Permission이므로 누구도 접근할 수 없다는 결론을 내릴 수 있다. 따라서, 전체 권한 체크 과정은 종결되고, "DENIED"라는 결과가 리턴된다.

"user1"이 "Administrator" Role에 매핑되어 있지만, "Administrator" RolePermission이 Excluded Permission이므로 "user1"을 포함한 누구도 "Administrator" Role에 접근할 수 없게 된다. 이는 Excluded Permission이 Unchecked Permission과 Checked Permission보다 우선순위가 높기 때문이다.

위에서 설명한 대로 차근 차근 과정을 따라가 보면 다음 권한 체크 질의가 어떻게 풀릴지 알 수 있을 것이다.

Principal	Operation	Context	Outcome
user1	JNDI:lookup	A	GRANTED
user1	JNDI:modify	A	DENIED
user1	JEUS:boot	A	DENIED
user1	JEUS:boot	B	GRANTED

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

1.4.4. 보안 감사

보안 감사는 일반적으로 보안과 관련된 이벤트(인증 실패, 런타임 예외 발생)를 포착하여 적절히 처리함으로써 전반적인 보안 품질을 향상시키는 것이다.

JEUS 보안 시스템은 보안 이벤트를 기반으로 하는 단순하지만 유연한 보안 감사 메커니즘을 특징으로 한다. 주요한 이벤트가 발생할 때마다(인증 실패, 권한 부여 실패, 관심도가 높은 기타 이벤트) 이벤트는 미리 등록된 이벤트 핸들러에 포착된다. 커스텀 핸들러 구현 클래스는 이에 적절하게 대응한다.

예를 들어 한 번에 연속적으로 너무 많이 인증에 실패할 경우 해당 Subject의 Credential에 Lock을 걸도록 할 수 있다.

1.4.5. 서비스와 SPI

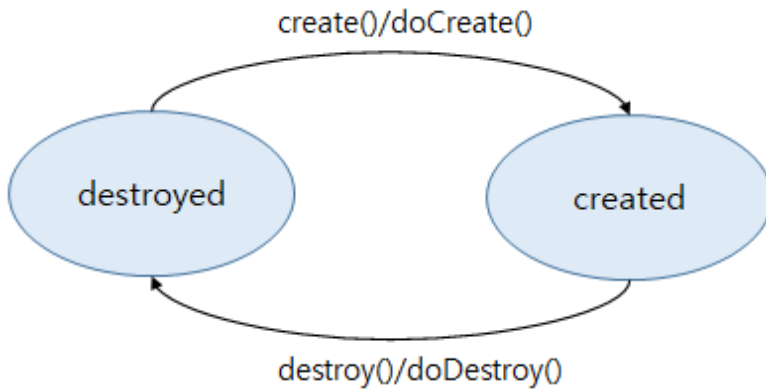
보안 시스템에서 실제 보안 기능을 구현하고 있는 클래스를 서비스라고 한다. 즉, 서비스는 특정 보안 기능(인증, 권한 부여, 네트워킹, 기타)을 제공하는 SPI를 구현한 클래스이다.

SPI는 jeus.security.spi 패키지에 포함되어 있는 추상 클래스이다. 커스텀 보안 기능을 구현하기 위해 SPI 클래스를 다양하게 확장하고 구현할 수 있다. SPI를 확장한 서브 클래스가 바로 서비스이다.

보안 시스템에서 서비스 인스턴스는 각각이 특정 보안 기능을 제공하는 독립적인 실체로 간주된다. 그러나 종종 서비스는 다른 SPI를 호출하기도 하기 때문에 서비스 간의 어느 정도 의존성은 존재한다. 서비스를 초기화하기 위해 모든 서비스는 "key-value"로 이루어진 한 쌍의 속성값을 전달받는다. 이러한 데이터는 설정 파일을 사용해서 저장해둘 수도 있다.

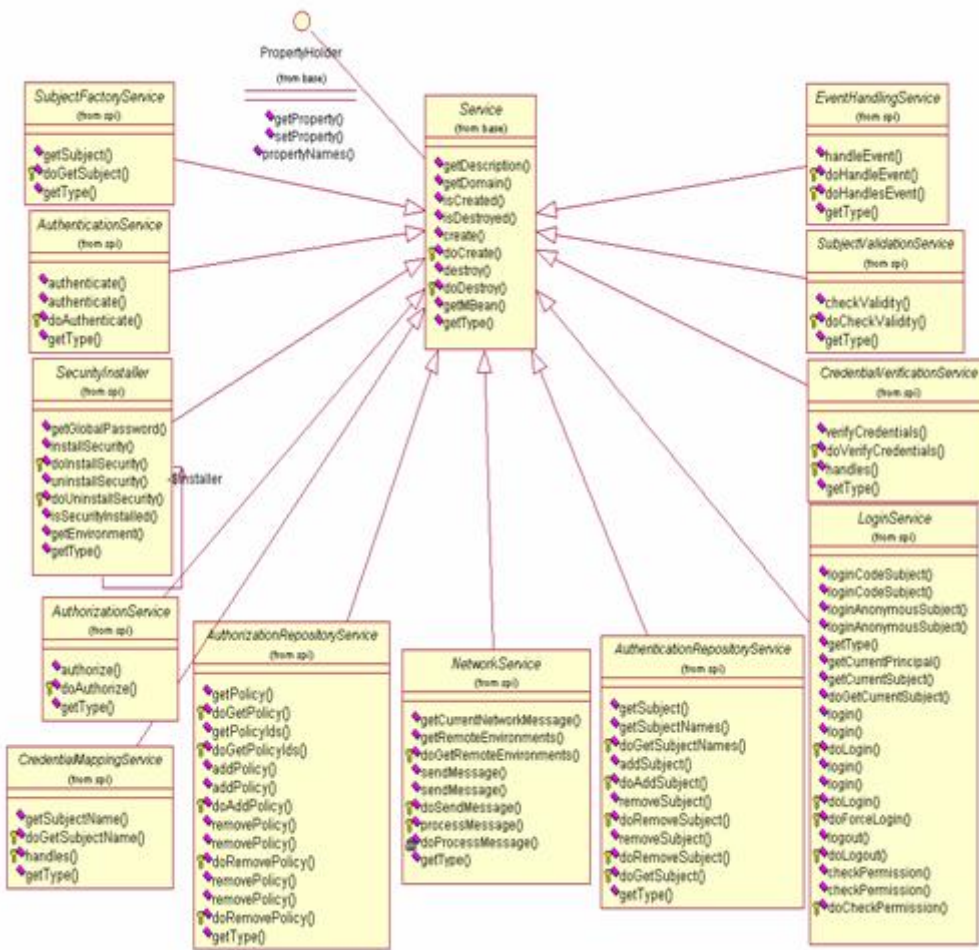
모든 서비스는 선택사항으로 JEUS 관리 시스템에 보고할 JMX Bean을 정의하기도 한다. JMX MBean은 보통 Service.getMBeanInfo() 메소드로부터 리턴받은 MBeanInfo를 기반으로 생성된다.

또한 모든 서비스는 생성(created)과 소멸(destroyed) 상태를 가지고 있다. 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.



서비스의 2가지 상태

다음은 jeus.security.spi 패키지에 포함되어 있는 다양한 SPI 클래스와 서비스 클래스들을 클래스 다이어그램을 사용해 보여주고 있다.



서비스 클래스와 SPI 서브 클래스들

런타임에 서비스 인스턴스는 SecurityInstaller 싱글톤 클래스에 의해 생성된다. 단순히 SecurityInstaller를 구현한다면 서비스를 프로그램에서 직접 생성하는 코드만 작성하면 되겠지만, 더 유연하고 정교하게 서비스를 생성하는 방법이 있다. 서비스 클래스명과 속성값을 특정 설정 파일에 저장해 놓고 파일로부터 설정 정보를 읽어들이어 서비스 인스턴스를 만들고 초기화하는 방법이다. 디폴트 SecurityInstaller 구현 클래스는 후자의 접근법을 도입하였으므로 쉽고 유연한 방법으로 서비스를 보안 시스템에 추가할 수 있다.



단순히 JEUS 보안 시스템 기능만을 사용할 때에는 서비스나 SPI 아키텍처를 모두 이해할

필요는 없다.

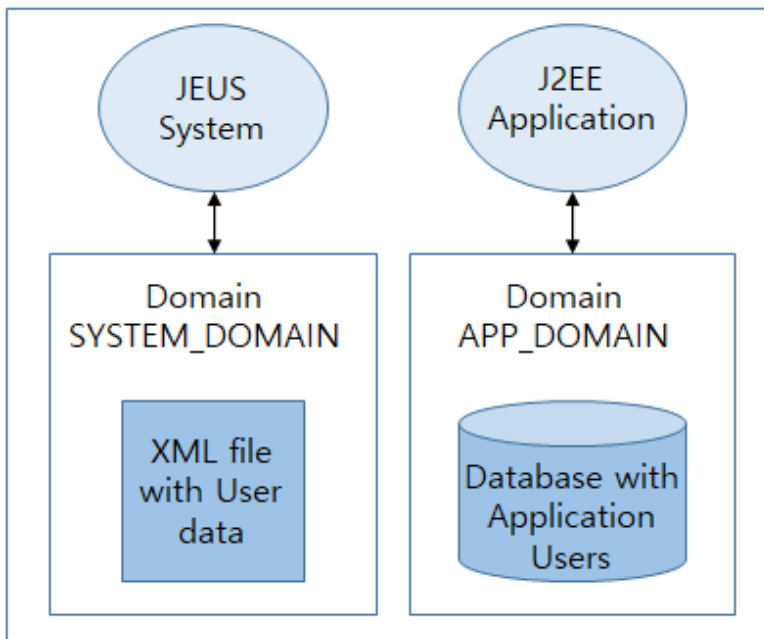
1.4.6. 도메인

보안 도메인(Security Domain)은 보안 서비스 인스턴스의 집합이라 할 수 있다. 하나의 보안 시스템에 여러 개의 도메인이 동시에 있을 수 있다. 도메인 개념의 배경에는 서로 다른 애플리케이션이나 JEUS 서버 시스템이 각각 별도의 보안 서비스를 사용하도록 하는 데 있다. 도메인은 보안 서비스를 각 애플리케이션별로 분리하는 역할을 한다.

예를 들어서 JEUS 시스템만 사용하는 특별한 도메인을 설정할 수 있다. JEUS가 사용하는 도메인을 SYSTEM_DOMAIN이라고 한다. 이 도메인은 JEUS를 관리하는 데 사용하는 Subject와 Policy를 포함하고 있다. SYSTEM_DOMAIN에 서버를 부트 또는 다운할 수 있는 "administrator"라는 메인 Subject에 대한 사용자 정보를 설정할 수 있을 것이다.

또 다른 성격의 도메인으로 deploy된 특정 Jakarta EE 애플리케이션에서만 사용하는 도메인을 설정할 수 있다. 이를 APP_DOMAIN이라 한다. APP_DOMAIN도 "administrator"라는 사용자 정보를 설정할 수 있지만, JEUS의 "administrator"와는 Permission이 다르다. 또한 도메인별로 Repository 메커니즘을 다르게 설정할 수 있다. 가령 SYSTEM_DOMAIN이 사용자 정보를 XML 파일에 저장하는데 반해, APP_DOMAIN은 원격지에 있는 데이터베이스를 사용해서 Subject에 대한 사용자 정의를 읽고 쓰도록 설정할 수 있다.

다음은 도메인 개념을 보여주고 있다.



다른 애플리케이션과 다른 Subject Repository를 사용한 2개의 도메인

도메인을 별도로 설정할지 여부는 선택사항이다. 만약 도메인을 별도로 설정하지 않는다면 SYSTEM_DOMAIN이 사용된다.



도메인명은 모두 대문자로 쓰고 "_DOMAIN"으로 끝나는 것이 보통이다. 이것은 반드시 지켜야 되는 것은 아니고 추천되는 명명법이다. 앞으로 이 문서에서 도메인이라 함은 모두 보안 도메인을 말하는 것이다. 보안 도메인은 Jakarta EE 서버 도메인과는 상관없는 별개의 개념이다.

다음과 같이 JEUS는 기본적으로 SYSTEM_DOMAIN과 SHARED_DOMAIN 2개의 도메인을 사용한다.

- SYSTEM_DOMAIN

표준 관리 툴을 사용하여 JEUS 서버를 관리하는 데 사용되는 도메인으로 기동과 종료같은 JEUS 서버를 관리하는 "administrator" 계정과 Permission을 포함하고 있다. 기본적으로는 Jakarta EE 애플리케이션도 이 도메인을 사용하며 설정을 통해 변경 가능하다.

- SHARED_DOMAIN

특별한 도메인으로 SHARED_DOMAIN에 설정된 보안 서비스는 다른 모든 도메인과 공유된다. 따라서, SHARED_DOMAIN은 모든 다른 도메인에서도 공통적으로 적용 가능한 보안 서비스를 포함하고 있어야 한다.

애플리케이션에 JEUS 시스템에서와 다른 보안 서비스를 사용하려면 별도의 도메인을 생성하고 domain.xml에 해당 내용을 적어 주어야 한다. 보안 도메인의 생성 및 설정은 [보안 시스템 설정](#)을 참고한다.

1.5. 성능 향상과 수준 향상

기본적으로 성능과 보안은 항상 타협의 대상이다. 보안이 강화되면 성능이 떨어지는 경향이 있기 때문에 보안 시스템이 가져야 하는 특징과 충돌되는 부분도 있다. 보안과 성능의 수준을 조절하여 각 업무의 특성에 따라 합리적으로 설정한다.

1.5.1. 성능 향상

본 절에서는 보안 시스템을 사용하면서 JEUS 서버 전체에 미치는 성능의 영향력을 최소화시키는 방법을 설명한다.

보안 시스템의 전반적인 성능을 향상시키는 방법은 다음과 같다.

- JEUS 도메인에서 Secured Connection을 사용하지 않는다.

Secured Connection은 일반 소켓에 암호화된 패킷을 담아 사용하게 된다. 그러므로 일반 소켓 통신을 할 때보다 더 많은 데이터를 전송하게 되어 네트워크를 많이 사용하게 된다.

JEUS 도메인에서는 Secured Connection을 사용하지 않도록 하는 것이 성능을 향상시킨다.

- JEUS SecurityManager를 사용하지 않는다.

SecurityManager를 사용하지 않으면 불필요한 연산을 수행하지 않게 된다. Security를 사용하지 않는다는 것은 Authorization을 사용하지 않는다는 의미이다(Authentication은 반드시 사용해야 한다).

- Non Blocking I/O를 사용한다.

Non-Blocking I/O를 사용하여 네트워크의 블록 현상을 줄인다. Non Blocking I/O는 네트워크를 통하여 I/O가 발생할 때 다음 I/O가 발생하기를 기다리는 블록(Block) 현상을 없앤 것이다.

따라서 각 커넥션마다 I/O를 위한 Thread가 별도로 필요하지 않기 때문에 CPU 사용량이나 FD의 개수도 줄게 된다. Non Blocking I/O는 JEUS에서 기본으로 사용하고 있다. 그러므로 Blocking I/O보다 높은 성능을 낸다.

1.5.2. 수준 향상

본 절에서는 성능을 희생하더라도 보안 시스템에서 전반적인 보안 수준을 향상시키는 방법에 대해 설명한다.

JEUS의 전반적인 보안 수준을 향상하기 위한 방법은 다음과 같다.

- 전역 시스템 패스워드를 설정한다.
- JEUS 보안 도메인에서 Secured Connection을 사용한다.

네트워크 보안 서비스가 Secured Connection(보통 SSL/TLS로 보호되는 커넥션)을 이용하도록 설정한다. 이 설정의 목적은 누군가가 네트워크를 도청하고 있다가 민감한 정보를 가져가는 것을 방지하는 것이다. 디폴트 네트워크 서비스를 사용할 때 Secured Connection을 설정하는 방법은 [참고 자료](#)를 확인한다.

- 저장된 Subject와 Policy를 권한이 없는 접근으로부터 보호한다.

Subject에 대한 사용자 정보와 Policy 저장소를 권한이 없는 접근으로부터 보호하는 것은 필수적인 일이다. 이는 사용하는 저장소의 종류에 따라 다음과 같이 방법이 달라진다.

- 데이터베이스를 사용하고 있다면 Subject에 대한 사용자 정보와 Policy를 저장하고 있는 테이블은 데이터베이스 측에 설정된 인증과 권한 부여 과정을 통해 보호될 수 있다. 또한 JEUS 보안 저장소와 데이터베이스 간의 커넥션을 SSL 등을 사용해서 보호하도록 한다. 설정하는 방법은 해당 데이터베이스 문서를 참고한다.
- 보안 관련 정보를 accounts.xml, policies.xml과 같이 XML 파일에 저장해 두었다면, 이러한 파일에 접근할 수 있는 Permission을 별도로 생성하고, JEUS 보안 시스템과 administrators만이 이 파일에 접근할 수 있도록 해야 한다. 만약 파일 저장소에 상관없이 암호화를 지원한다면 이를 사용한다. 어떻게 파일에 보안을 적용할 것인가에 대한 정보는 해당 저장소에 대한 문서를 참고한다. 이 파일에 대한 읽기, 쓰기 권한을 설정하는 것은 해당 운영체제의 매뉴얼을 참고한다.

- 보안 감사 메커니즘을 도입한다.

시스템에서 발생한 보안 이벤트를 로그에 기록하고, 주기적으로 로그 내용을 살펴보면 전체적인 보안 수준을 향상시킬 수 있다. JEUS 보안 감사 메커니즘은 보통 jeus.security.spi.EventHandling Service SPI 클래스를 사용해서 구현된다. 로그 파일을 권한이 없는 접근으로부터 보호해야 한다. 다양한 보안 감사 메커니즘에 대한 설명은 [참고 자료](#)를 참고한다.

- Java SE SecurityManager를 사용한다.

JEUS 시스템의 견고성을 높이고 악의적인 EJB와 서블릿 코드로부터 JEUS를 보호하기 위해서 Java SE SecurityManager를 설정한다. 자세한 내용은 [Java SE SecurityManager 설정](#)을 참고한다.

- 3rd-party 보안 메커니즘을 사용한다.

민감한 정보가 다루어지는 실제 환경(은행 업무 애플리케이션 등)에서는 JEUS 본래의 보안 시스템과 더불어 방화벽이나 VPNs(Virtual Private Networks)와 같은 추가적인 보안 요소들을 설치하는 것을 권장한다.

- 운영체제 자체에 보안을 설정한다.

JEUS가 동작하는 운영체제의 종류에 상관없이 운영체제를 보호하는 몇 가지 가이드 라인이 있다.

- JEUS가 설치된 머신들에 대한 물리적 접근을 제한한다. 예를 들면 머신들을 한 곳에 두고 자물쇠로 채운다.

- 신뢰가 확보된 관리자(administrator)만이 JEUS 파일과 프로세스에 접근할 수 있도록 프로세스 권한이나 파일 권한을 설정한다. 자세한 내용은 해당 운영체제 매뉴얼을 참고한다.
- 최신 보안 패치가 적용된 운영체제로 업그레이드한다.
- 주기적으로 안티 바이러스 소프트웨어를 작동시킨다.
- 모든 보안 관련 문서들을 안전하게 보관한다. 예를 들면 패스워드가 적힌 문서를 자물쇠가 달린 캐비닛에 보관한다.

2. 보안 시스템 설정

본 장에서는 JEUS에서 핵심적인 보안 시스템을 실제로 어떻게 설치하고 설정하는지를 설명한다. 설명한 서비스 이외의 서비스를 도메인에 설정하는 방법은 [참고 자료](#)를 참고한다.

2.1. 개요

본 절에서는 보안 시스템 설정과 관련된 기본적인 사항들을 간단하게 설명한다.

JEUS의 보안 시스템은 Security Installer를 통해 서버를 실행할 때 시작된다. Security Installer는 domain.xml에 정의된 정보를 바탕으로 보안 도메인을 만들고 보안 서비스를 시작하여 JEUS를 안전하게 보호한다. 디폴트 보안 시스템에서는 accounts.xml과 policies.xml에 정의된 사용자 계정과 보안 정책이 적용된다.

보안 도메인에 대해 다음의 2가지를 설정할 수 있다.

- 보안 도메인과 보안 서비스에 대한 정의
- 해당 도메인에 대한 계정 및 정책 설정

디폴트 보안 시스템의 설정 과정은 다음과 같다.

1. 보안 도메인들을 설정한다.
2. 각 보안 도메인별로 구성요소와 보안 서비스를 설정한다.
3. 각 도메인에 대한 Subjects(인증 데이터)의 사용자 정보를 설정한다.
4. 각 도메인에 대한 Policies(보안 정책 데이터)를 설정한다.
5. Subject와 Policy 이외의 추가 사항을 설정한다.
6. Java SE SecurityManager를 설정한다(선택적).
7. JACC Provider를 설정한다(선택적).

디폴트 보안 시스템 디렉터리

다음은 디폴트 보안 시스템의 디렉터리 구조이다. 각 디렉터리에는 보안 시스템에서 사용하는 설정 파일들이 나열되어 있다.

```
JEUS_HOME/domains/<domain name>
|--config
  |--security
    |--security.key
    |--policy
    |--security-domains.xml
    |--SYSTEM_DOMAIN
      |--accounts.xml
      |--policies.xml
```

security

다음은 각 디렉터리 및 파일에 대한 설명이다.

구분	설명
security.key	대칭키 암호화 알고리즘의 Key를 저장하는 파일이다. 최초로 암호화를 수행하면 파일이 생성된다.
policy	Java Permission 설정 파일이다. JEUS의 보안 시스템과는 별도로 Java SE Security Manager에서 사용된다.
security-domains.xml	JEUS의 보안 도메인에 대한 설정을 저장하는 파일이다.
SYSTEM_DOMAIN	JEUS 서버가 사용자 인증과 권한 체크를 위해 사용하는 도메인이다. 이 도메인은 JEUS를 기동하고, 종료하는 등의 Permission들과 JEUS 시스템 administrator 계정을 포함하고 있다. <ul style="list-style-type: none"> accounts.xml : 사용자 정보가 저장된다. policies.xml : 보안 정책 정보(권한 부여 데이터)가 저장된다.



애플리케이션에 JEUS 시스템과 다른 보안정책을 적용하려면 별도의 보안 도메인 디렉터리를 생성하고 관련 설정을 적용한 뒤 사용해야 한다.

2.2. 보안 도메인 설정

보안 도메인을 설정하기 위해서는 domain.xml 및 security-domains.xml을 직접 편집해야 한다. 사용자 계정 설정과 보안 정책 설정을 제외한 나머지 사항은 동적 변경이 불가능하다. 따라서 보안 도메인을 추가하거나, 보안 서비스 설정 등을 수정했을 경우 서버를 재부팅해야 수정된 사항이 적용됨에 유의한다.



보안과 관련된 모든 설정은 JEUS 도메인에 속한 모든 서버에 동일하게 적용된다. 따라서 사용자 계정 설정과 보안 정책 설정을 제외하고는 동작하고 있는 모든 서버를 재부팅해야 보안 설정 적용이 가능하다.

본 절에서는 XML 편집을 통해 보안 도메인을 정의하는 방법에 대해 설명한다.

2.2.1. XML 편집 설정

XML을 직접 편집하려면 JEUS_HOME/domains/<domain name>/config/domain.xml 및 JEUS_HOME/domains/<domain name>/config/security/security-domains.xml 내의 다음 태그를 이용하여 설정한다.

도메인에 대한 XML 파일 설정 방법은 JEUS_HOME/lib/schemas/jeus/supportLocale/ko 디렉터리 내의 jeus-domain.xsd와 jeus-security.xsd, security-domains.xsd 의 XML 스키마에 정의되어 있다. 상위 태그는 **<security-manager>**이며, 이 태그는 도메인 설정과 관련된 다음 하위 태그를 가진다. 하위 태그는 0개 이상씩 설정 가능하다.

보안 도메인 설정 : <domain.xml>

```
<security-manager>
```

```

<default-application-domain>SYSTEM_DOMAIN</default-application-domain>
<security-domain-names>
  <security-domain-name>SYSTEM_DOMAIN</security-domain-name>
  ...
</security-domain-names>
</security-manager>

```

다음은 설정 태그에 대한 설명이다.

- <security-manager>

JEUS에 등록할 보안 도메인들을 설정한다. <security-manager>에 대한 보안 설정 파일의 예제는 [보안 도메인 구성요소 설정](#)을 참고한다.

태그	설명
<connect-retries>	JEUS Security NetworkService에서 connection retry count 값을 정의한다. (기본값: 10)
<password-validator>	JEUS 계정의 암호를 지정할 때, 암호의 보안성을 높이기 위하여 암호 적합성 검사를 설정한다.
<default-application-domain>	Jakarta EE 애플리케이션에서 사용할 기본 보안 도메인의 이름을 정의한다. (기본값: SYSTEM_DOMAIN)

- <security-domain-names>

JEUS에서 사용할 보안 도메인들의 이름을 지정한다.

태그	설명
<security-domain-name>	Security 도메인 이름을 설정한다. Security 도메인 설정에서 name 값을 참조한 값이다.

보안 도메인 설정 : <security-domains.xml>

```

<security-domains>
  <security-domain>
    <name>SYSTEM_DOMAIN</name>
    <authentication />
    <authorization />
    ...
  </security-domain>
</security-domains>

```

다음은 설정 태그에 대한 설명이다.

- <security-domain>

JEUS에 등록할 보안 도메인들을 설정한다. <security-domain>에 대한 보안 설정 파일의 예제는 [보안 도메인 구성요소 설정](#)을 참고한다.

태그	설명
<name>	Security 도메인 이름을 설정한다.
<authentication>	Authentication 서비스를 정의한다.
<authorization>	Authorization 서비스를 정의한다.
<identity-assertion>	Identity Assertion 서비스를 정의한다.
<credential-mapping>	Credential Mapping 서비스를 정의한다.
<credential-verification>	Credential Verification 서비스를 정의한다.
<audit>	Audit 서비스를 정의한다.
<subject-validation>	Subject Validation 서비스를 정의한다.

2.2.2. 사용자 계정 및 보안 정책 설정

본 절에서 설명은 디폴트로 제공되는 XML을 이용한 설정에 대해 설명한다.

사용자 계정 및 보안 정책 설정은 XML뿐만 아니라 데이터베이스에 사용자 계정 정보와 보안 정책 정보를 저장하여 사용하는 것도 가능하며, 이에 대한 자세한 내용은 사용자 정보 설정에 대한 [데이터베이스 사용 설정](#)과 보안 정책 설정에 대한 [데이터베이스 사용 설정](#)을 참고한다.

다음은 사용자 계정 및 보안 정책을 설정하는 과정에 대한 설명이다.

1. 새로운 보안 도메인을 설정하려면 JEUS_HOME/domains/<domain name>/config/security 아래에 새로운 디렉터리를 생성해야 한다. 디렉터리 이름은 생성하려는 보안 도메인 이름과 일치해야 한다. 일반적으로 보안 도메인 이름은 모두 대문자로 쓰고 "_DOMAIN"으로 끝맺는다. 예를 들면 "DEFAULT_APPLICATION_DOMAIN"으로 명명할 수 있다.

DEFAULT_APPLICATION_DOMAIN 도메인을 생성하기 위해 명령 프롬프트에서 다음 명령어를 실행한다.

```
$ mkdir ${JEUS_HOME}/domains/domain1/config/security/DEFAULT_APPLICATION_DOMAIN
```



domain1은 실제 JEUS 도메인 이름으로 대체되어야 한다.

2. 새로운 도메인 디렉터리를 생성한 후에 디렉터리 내에 몇 가지 설정 파일들을 생성한다.

가장 좋은 방법은 기존 도메인 디렉터리의 설정 파일들을 그대로 복사해서 필요에 따라 설정 파일을 변경하는 것이다(다음의 명령어는 모두 한 라인에 쓴다). 복사된 설정 파일에 대한 설명은 다음 절에서 설명한다.

```
$ cp ${JEUS_HOME}/domains/domain1/config/security/SYSTEM_DOMAIN/*. *
  ${JEUS_HOME}/domains/domain1/config/security/DEFAULT_APPLICATION_DOMAIN
```

JEUS_HOME/domains/<domain name>/security 아래에 SYSTEM_DOMAIN이 이미 디폴트로 존재한다.

SYSTEM_DOMAIN은 JEUS 서버가 사용자 인증과 권한 체크를 위해 사용하는 도메인이다. 이 도메인은 JEUS를 기동하고, 종료하는 등의 Permission들과 JEUS 시스템 "administrator" 계정을 포함하고 있다.

3. 새로운 보안 도메인 이름을 JEUS_HOME/domains/<domain name>/config/domain.xml 및 JEUS_HOME/domains/<domain name>/config/security/security-domains.xml에 각각 추가한다. domain.xml의 경우는 <security-manager><security-domain-names>에 추가해야 하고, security-domains.xml의 경우는 <security-domains><security-domain>에 추가한다.
4. 새로 생성된 도메인은 JEUS를 다시 시작해야 적용된다.

보안 도메인 SYSTEM_DOMAIN은 domain.xml의 설정과는 관계없이 항상 포함된다. 기본적으로 JEUS_HOME/domains/<domain name>/config/security 경로 아래에 도메인과 같은 이름의 디렉토리를 생성하여 각 도메인별로 정의할 Repository에 대한 계정 정보(account.xsd)와 보안 정책 정보(policies.xsd)를 정의한다. 기타 도메인별 Security Service 등록은 domain.xml에 한다.

2.3. 보안 도메인 구성요소 설정

본 절에서는 보안 서비스를 제외한 보안 도메인의 구성요소들을 설정하는 방법에 대해 설명한다.

2.3.1. XML 편집 설정

각 보안 도메인에서 로딩되는 보안 서비스는 **domain.xml** 및 **security-domains.xml**에 설정되어 있다.

서비스에 대한 XML 파일 설정 방법은 JEUS_HOME/lib/schemas/jeus/supportLocale/ko 디렉토리 내의 jeus-security.xsd 및 security-domains.xsd의 XML 스키마에 정의되어 있다. <security-domains> 태그의 하위에는 서비스 설정과 관련된 <security-domain>을 설정한다. <security-domain>은 1개 이상 설정이 가능하고 각 태그는 JEUS에서 사용하는 보안 도메인을 정의한다.

보안 도메인 설정 : <domain.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<domain version="9.0" xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    . . .
    <security-manager>
        <default-application-domain>DEFAULT_APPLICATION_DOMAIN</default-application-domain>
        <security-domain-names>
            <security-domain-name>SYSTEM_DOMAIN</security-domain-name>
            <security-domain-name>DEFAULT_APPLICATION_DOMAIN</security-domain-name>
            . . .
        </security-domain-names>
    </security-manager>
    . . .
</domain>
```

보안 시스템 서비스 설정 : <security-domains.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<security-domains xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9.0">
    <security-domain>
        <name>SYSTEM_DOMAIN</name>
```

```

.....
</security-domain>
<security-domain>
  <name>DEFAULT_APPLICATION_DOMAIN</name>
</security-domain>
</security-domains>

```



<security-domain>에서 기타 다른 Service Provider에 대한 정보를 설정하지 않고 <name>에 대한 정보만 설정하면 해당 도메인에 기본으로 제공되는 보안 서비스들이 동작하게 된다.

다음은 <security-domain>의 하위 태그에 대한 설명이다.

- <name> (필수)

보안 도메인에 대한 이름이다.

- <cache-config> (0개 이상, 선택)

해당 도메인에 적용된 보안 Repository Service에서 적용할 Cache 정책값을 정의한다.

태그	설명
<min>	Repository Service에 적용할 Cache entry 최소 size 값을 설정한다.
<max>	Repository Service에 적용할 Cache entry 최대 size 값을 설정한다.
<timeout>	Repository Service에 적용할 Cache entry의 timeout 값을 설정한다.

- <keystore-config> (0개 이상, 선택)

해당 도메인의 보안 서비스에 적용할 Keystore 파일에 정보값을 정의한다.

하위의 설정값이 없는 경우 -Djeus.ssl.* 또는 -Djavax.net.ssl.* 설정이 되어 있는지 확인해 보고 해당 값이 없을 경우에는 기본값이 적용된다.

태그	설명
<keystore-path>	Keystore 파일 경로를 설정한다. (예: JEUS_HOME/domains/<domain name>/config/security/keystore)
<keystore-alias>	Keystore 파일의 KeyEntry 타입의 인증서가 여러 개인 경우 명시적으로 alias 값으로 해당 서버 인증에 필요한 인증서를 가리키도록 한다.
<keystore-password>	Keystore 파일에 대한 패스워드를 설정한다. (기본값: changeit)
<keystore-keypassword>	Keystore 파일에 대한 Keypassword를 설정한다. (기본값: <keystore-password> 값과 동일)

태그	설명
<truststore-path>	Truststore 파일 경로를 설정한다. (예: JEUS_HOME/domains/<domain name>/config/security/truststore)
<truststore-password>	Truststore 파일에 대한 패스워드를 설정한다. (기본값: changeit)

2.4. 보안 서비스 설정

JEUS 보안 시스템은 플러그(plug) 형태의 인증(Authentication) 서비스와 권한 확인(Authorization) 서비스를 지원한다. 본 절에서는 보안 서비스를 포함하여 보안 도메인의 구성요소들을 XML 편집을 통해 설정하는 방법에 대해 설명한다.

2.4.1. XML 편집 설정

각 보안 도메인에서 로딩되는 보안 서비스는 **security-domains.xml** 내에 다음과 같이 설정되어 있다.

다음은 보안 설정 파일의 예제이다.

보안 시스템 서비스 설정 : <security-domains.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<security-domains xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9.0">
  <security-domain>
    <name>SYSTEM_DOMAIN</name>
    <authentication>
      <repository-service>
        <xml-file-repository>
          <config-file>
            <filename>accounts.xml</filename>
            <filepath>
              ${JEUS_HOME}/domains/domain1/config/security/
            </filepath>
          </config-file>
        </xml-file-repository>
      </repository-service>
    </authentication>
  </security-domain>
  <security-domain>
    <name>DEFAULT_APPLICATION_DOMAIN</name>
  </security-domain>
</security-domains>
```

다음은 <security-domain>의 하위 태그에 대한 설명이다.

- <name> (필수)

보안 도메인에 대한 이름이다.

- <authentication> (0개 이상, 선택)

해당 도메인에 적용할 인증 서비스를 정의한다.

- <repository-service>

인증을 위한 사용자 정보 저장소 타입에 따른 서비스를 정의한다.

태그	설명
<xml-file-repository>	사용자 정보가 xml 파일에 정의된 경우 설정한다.
<database-repository>	사용자 정보가 데이터베이스에 정의된 경우 설정한다.
<custom-repository>	jeus.security.spi.AuthenticationRepositoryService SPI를 상속하여 사용자 정보를 로딩하는 방식을 확장 Repository Service를 구현하여 적용하는 경우 설정한다.

- <jaas-login-config>

해당 도메인에 적용할 JAAS 서비스를 등록한다.

- <callback-handler-class> : JAAS Callback Handler Factory 클래스 이름을 정의한다.
- <login-module> : LoginModule 관련 내용을 설정한다.

태그	설명
<login-module-classname>	LoginModule을 implements한 패키지를 포함한 클래스 이름을 정의한다.
<control-flag>	다음 4가지 속성들 중 하나를 정의하여 전반적으로 Authentication Stack에 대하여 조정하는 속성을 정의한다. <ul style="list-style-type: none"> ◦ required ◦ requisite ◦ sufficient ◦ optional
<option> (0개 이상)	LoginModule 초기화하는 경우 적용할 속성값을 정의한다.

- <custom-authentication-service>

jeus.security.spi.AuthenticationService SPI를 상속하여 기본으로 제공되는 인증 서비스를 확장해서 적용하고 싶은 경우 정의한다. 설정은 "[Custom 보안 서비스](#)" 설명을 참고한다.

- <authorization> (0개 이상, 선택)

해당 도메인에 적용할 권한 부여 서비스를 정의한다.

- <repository-service>

권한 부여를 위한 정책 정보 저장소 타입에 따른 서비스를 정의한다.

태그	설명
<xml-file-repository>	정책 정보가 XML 파일에 정의된 경우 설정한다.
<database-repository>	정책 정보가 데이터베이스에 정의된 경우 설정한다.
<custom-repository>	jeus.security.spi.AuthorizationRepositoryService SPI를 상속하여 정책 정보를 로딩하는 방식을 확장 Repository Service를 구현하여 적용하는 경우 설정한다.

◦ <custom-authorization-service>

jeus.security.spi.AuthorizationService SPI를 상속하여 기본으로 제공되는 인증 서비스를 확장 적용하고 싶은 경우 정의한다. 설정은 "[Custom 보안 서비스](#)" 설명을 참고한다.

◦ <jacc-service>

JACC 2.0 기반의 권한 부여 서비스를 적용하는 경우 설정한다.

• <identity-assertion> (0개 이상, 선택)

해당 도메인에 적용할 IdentityAssertion 서비스를 정의한다.

◦ <default-identity-assertion-service>

JEUS Security Framework에서 제공하는 기본 Identity Assertion Service를 정의한다.

- <x509-identity-assertion> : X509Certificate Token에 대한 Identity Assertion Service를 지원한다.

태그	설명
<config-file>	X509Certificate Token에 대한 사용자 매핑을 하기 위해서 정보를 정의한 파일의 위치를 지정한다. (기본값: DOMAIN 폴더 하위에 위치한 user-cert-map.xml)
<default-user-mapper>	X509Certificate Token 값에 attribute type(cert-attr-type) 값 또는 attribute type(attribute-type), attribute value에 대한 delimiter 값(attribute-value-delimiter)을 정의한다.

◦ <custom-identity-assertion-service>

jeus.security.spi.IdentityAssertionService SPI를 상속한 Identity Assertion Service를 확장 구현하여 정의한다. 설정은 "[Custom 보안 서비스](#)" 설명을 참고한다.

• <credential-mapping> (0개 이상, 선택)

해당 도메인에 적용할 CredentialMapping Service를 정의한다.

- <default-credential-mapping-service>

JEUS Security Framework에서 제공하는 기본 Credential Mapping Service를 지원한다.

- <x509-credential-mapping> : X509Certificate에 대한 Credential Mapping Service를 지원한다.

태그	설명
<truststore-path>	현 도메인에 적용할 Truststore 파일에 대한 경로를 정의한다.
<truststore-password>	현 도메인에 적용된 Truststore파일 대한 패스워드를 정의한다.

- <custom-credential-mapping-service>

jeus.security.spi.CredentialMappingService SPI를 상속한 Credential Mapping Service를 확장 구현하여 정의한다. 설정은 "[Custom 보안 서비스](#)" 설명을 참고한다.

- <credential-verification> (0개 이상, 선택)

해당 도메인에 적용할 Credential Verification Service를 정의한다.

- <default-credential-verification-service>

JEUS Security Framework에서 제공하는 기본 Credential Verification Service 지원한다.

태그	설명
<password-verification>	PasswordFactory 클래스에 대한 검증 서비스를 정의한다.
<jeus-certificate-verification>	X509Certificate에 대한 검증 서비스를 정의한다.

- <custom-credential-verification-service>

jeus.security.spi.CredentialVerificationService SPI를 상속한 Credential Verification Service를 확장 구현하여 정의한다. 설정은 "[Custom 보안 서비스](#)" 설명을 참고한다.

- <audit> (0 개 이상, 선택)

JEUS Security Framework에서 발생하는 이벤트에 대한 정보를 수집할 수 있도록 해당 도메인에 적용할 EventHandlingService를 정의한다.

태그	설명
<default-audit-service>	<p>해당 이벤트 정보를 기록하는 파일 경로와 이벤트 로그 레벨을 정의한다.</p> <ul style="list-style-type: none"> • config-file: 로그 파일 경로 • audit-level: 로그 레벨

태그	설명
<custom-audit-service>	jeus.security.spi.EventHandlingServiceService SPI를 상속한 Audit Service를 확장 구현하여 정의한다. 설정은 " Custom 보안 서비스 " 설명을 참고한다.

Custom 보안 서비스

다음은 Custom 보안 서비스를 설정하는 방법이다. JEUS에서 제공하는 보안 서비스 종류와 Custom 보안 서비스의 개발과 관련된 사항은 [Custom 보안 서비스 개발](#)을 참고한다.

- <classname> (필수)

Custom 보안 서비스를 구현한 Java 클래스명이다. 이 클래스는 파라미터가 없는 디폴트 public 생성자를 가지고 있어야 하며, jeus.security.spi 패키지의 SPI 클래스를 상속받거나, jeus.security.base.Service 클래스를 직접 상속해야 한다.

- <property> (0개 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 초기화하는 데 사용된다.

다음 2개의 하위 태그를 가진다.

태그	설명
<name>	속성명을 설정한다.
<value> (선택)	속성명에 해당하는 String 속성값을 설정한다.

2.5. 보안 시스템 사용자 정보 설정

디폴트 보안 설정에서 사용자 데이터는 **accounts.xml** 파일에서 읽어 들인다.

다음은 파일 저장 경로이다.

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/accounts.xml
```

여기서 <domain name>은 도메인의 이름이고, <security domain name>은 사용자가 관리될 보안 도메인의 이름을 나타낸다.

2.5.1. XML 편집 설정

accounts.xml의 XML 스키마는 accounts.xsd이며, JEUS_HOME/lib/schemas/jeus/supportLocale/ko 경로에 있다. accounts.xml 파일에는 최상위 태그 <accounts> 하위에 <users> 및 <groups> 태그가 있으며 0개 이상의 <user> 및 <group> 태그가 포함되어 있다. 각각은 사용자와 그룹을 나타낸다.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>user1</name>
      <password>{AES}mnG6ItxF0/WQlE2YzIZ7sA==</password>
      <group>group1</group>
    </user>
    <user>
      <name>user2</name>
      <password>{DES}7dJ0KDTGQNpSnQAPYBNnmA==</password>
    </user>
    <user>
      <name>user3</name>
      <password>{DES}7dJ0KDTGQNpSnQAPYBNnmA==</password>
      <group>nested_group</group>
    </user>
    <user>
      <name>user4</name>
      <password>{SEED}d12EePMAcnxPYbIyknuZkA==</password>
      <group>nested_group</group>
    </user>
  </users>
  <groups>
    <group>
      <description>Group1</description>
      <name>group1</name>
      <subgroup>nested_group</subgroup>
    </group>
    <group>
      <description>For NestedGroup</description>
      <name>nested_group</name>
    </group>
  </groups>
</accounts>
```

다음은 설정 태그에 대한 설명이다.

- <user>

각 <user> 태그는 다음과 같은 하위 태그를 가지고 있다.

태그	설명
<description> (선택)	<user>에 대한 설명이다. (문자열)
<name> (필수)	<user>에 대한 이름을 나타낸다. (예: 사용자명, 사용자 ID)
<password> (0개 이상, 선택)	<p><user>에 대한 패스워드를 나타낸다.</p> <p><password>는 어떤 특정 알고리즘을 사용하여 인코딩된 값을 정의한다. <password> 태그에 적용된 알고리즘 또는 인코딩 방식을 '{}' 블록 사이에 기입하도록 한다. 인코딩 방식에 대한 자세한 내용은 패스워드 보안 설정을 참고한다.</p>

태그	설명
<group> (0개 이상, 선택)	<user>가 포함된 그룹을 나타낸다. 한 사용자는 여러 그룹에 포함될 수 있으며, 다음의 <group> 태그에서 정의된 그룹을 지정해야 한다.

- <group>

각 <group> 태그는 다음과 같은 하위 태그를 가지고 있으며 그룹을 정의한다.

태그	설명
<description> (선택)	<group>에 대한 설명이다. (문자열)
<name> (필수)	<group>에 대한 이름을 나타낸다.
<subgroup> (0개 이상, 선택)	해당 group role을 동일하게 부여할 수 있도록 nested하게 속한 그룹 이름들을 정의한다.

2.5.2. 데이터베이스 사용 설정

본 절에서는 JEUS가 정의한 데이터베이스 테이블을 이용한 Authentication을 설정하는 방법에 대해서 설명한다.

다음 예제는 JEUS에 정의한 데이터소스를 이용할 때 설정하는 방법이다.

데이터베이스를 이용한 사용자 설정 : <security-domains.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<security-domains xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9.0">
    . . .
    <security-domain>
        <name>MY_DOMAIN</name>
        <authentication>
            <repository-service>
                <database-repository>
                    <datasource-id>auth</datasource-id>
                </database-repository>
            </repository-service>
        </authentication>
        . . .
    </security-domain>
    . . .
</security-domains>
```

위의 예제에서 <datasource-id>에는 Authentication에 사용할 security-domains.xml에 등록한 데이터소스의 ID를 명시해야 한다. 또 다른 방법으로 DriverManager를 이용해 JEUS에서 제공하는 JDBC를 이용하지 않고 직접 DB와 통신하면서 Authentication을 이용할 수 있는데, 위의 예제에서 <datasource-id> 태그 부분을 다음과 같이 변경한다.

다음 예제는 JEUS JDBC를 이용하지 않을 때 데이터베이스 Repository를 설정하는 방법이다.

JEUS JDBC를 사용하지 않는 경우 설정 : <security-domains.xml>

```
<security-domain>
```

```

...
<repository-service>
  <dbdriver-config>
    <vendor>oracle</vendor>
    <driver>oracle.jdbc.OracleDriver</driver>
    <url>jdbc:oracle:thin:@127.0.0.1:1521:ORA9I</url>
    <username>scott</username>
    <password>{base64}dGlnZXI=</password>
  </dbdriver-config>
</repository-service>
...
</security-domain>

```



이 방식을 이용할 경우 커넥션을 맺고 끊는 것이 매우 잦기 때문에 성능상 문제가 있을 가능성이 크므로 JEUS JDBC(<datasource-id>)를 이용하는 방법을 권장한다.

데이터베이스에서 사용되는 테이블은 다음과 같이 구성된다.

jeus_group_members		
PK	domain	VARCHAR(30)
PK	groupname	VARCHAR(100)
PK	username	VARCHAR(30)

jeus_role_users		
PK	domain	VARCHAR(30)
PK	username	VARCHAR(30)
	password	VARCHAR(100)

jeus_groups		
PK	domain	VARCHAR(30)
PK	groupname	VARCHAR(100)
	subgroups	VARCHAR(100)
	description	VARCHAR(100)

Subject에 대한 사용자 정보를 저장하기 위한 데이터베이스 테이블 구조

데이터베이스에 테이블이 존재하지 않을 경우 JEUS_HOME/templates/security/dbrealm.sql.template 파일을 이용해서 DB에 테이블을 생성하면 사용자명이 "jeus", 패스워드가 "jeus"인 사용자가 최초로 생성된다.

2.5.3. 패스워드 보안 설정

본 절에서는 패스워드 보안 설정에 대한 사항 중 패스워드의 적합성 검사와 암호화 알고리즘, 그리고 SecretKey 파일 관리방법에 대해서 설명한다.

패스워드의 적합성 검사

사용자가 신규 계정을 생성하기 위해 패스워드를 입력하거나 기존 패스워드를 변경할 경우 패스워드의 적합성을 검사하여 제약을 줄 수 있다. 일반적으로 사용되는 제약 사항(알파벳과 숫자, 특수문자의 사용 여부나 아이디와 패스워드를 같게 하면 안되는 규칙 등등)과 같은 방식을 이용하여 패스워드의 보안을 강화할 수 있다.

Domain administrator는 기본적으로 제공되는 Default Password Validator나 사용자가 직접 구현하는 Custom Password Validator 중에 하나를 선택하여 도메인 전체에 적용할 수 있다. 각각의 기능을 사용하기 위해 사용자는 jeusadmin의 명령어를 통해 설정을 추가, 변경 및 삭제 할 수 있다. 사용자의 설정 정보는 **domain.xml**에 저장된다.

• Default Password Validator 설정

jeusadmin에서 **modify-default-password-validator**, **show-default-password-validator** 명령어를 통해 설정할 수 있으며, 명령어에 대한 정보는 JEUS Reference 안내서의 "Part II. 콘솔 명령어와 툴"을 참고한다.

```
[MASTER]domain1.adminServer>modify-default-password-validator -min 4 -max 10
Default password validator is updated successfully.
Check the results using "show-default-password-validator or modify-default-password-validator".
[MASTER]domain1.adminServer>show-default-password-validator
=====
+-----+-----+
|               property               | value |
+-----+-----+
| min length                           |    4  |
| max length                           |   10  |
| include special characters            | false |
| include digit characters              | false |
| include capital characters            | false |
| include small characters              | false |
| exclude user id                      | false |
+-----+-----+
=====
```

다음은 Default Password Validator을 사용할 경우 설정 항목에 대한 설명이다.

항목	설명
min	패스워드의 최소 글자 수를 설정한다. (범위: 1 ~ 255, 기본값: 1)
max	패스워드의 최대 글자 수를 설정한다. (범위: 1 ~ 255, 기본값: 255)
special	패스워드에 특수문자가 반드시 들어가게 설정한다.
digit	패스워드에 숫자가 반드시 들어가게 설정한다.
capital	패스워드에 대문자가 반드시 들어가게 설정한다.
small	패스워드에 소문자가 반드시 들어가게 설정한다.
excludeId	패스워드에 아이디가 포함되지 못하게 설정한다.

이렇게 설정한 Default Password Validator 값은 **domain.xml**에 아래와 같은 형식으로 저장된다.

Default Password Validator로 패스워드 적합성 검사 설정 : <domain.xml>

```
<domain>
. . .
<security-manager>
. . .
<password-validator>
```

```

        <default-password-validator>
            <minLength>4</minLength>
            <maxLength>10</maxLength>
            <force-special-character>true</force-special-character>
            <force-digit>true</force-digit>
            <force-capital-letter>true</force-capital-letter>
            <force-small-letter>true</force-small-letter>
            <deny-username>true</deny-username>
        </default-password-validator>
    </password-validator>
    . . .
</security-manager>
    . . .
</domain>

```

Default Password Validator의 설정을 마치면 변경 사항은 모두 동적으로 반영된다. 이후부터 새 사용자의 패스워드를 입력하거나 기존 패스워드를 변경하려고 할 때 설정된 Default Password Validator의 Password Validation이 적용된다.

• Custom Password Validator 설정

Custom Password Validator를 사용하면 사용자 맞춤형 Password Validation을 할 수 있다. JEUS는 jeus.util.PasswordValidator 인터페이스를 제공하며, 사용자는 해당 인터페이스를 직접 implement하여 원하는 기능을 담은 클래스를 구현한 뒤, jar 파일로 패키징하여 "DOMAIN_HOME/lib/application"에 넣어야 한다.

사용자가 implement하여 구현해야 하는 jeus.util.PasswordValidator 인터페이스는 다음과 같다.

```

public interface PasswordValidator {
    boolean validatePassword(String id, String password);
}

```

PasswordValidator 인터페이스를 구현한 클래스의 validatePassword(String id, String password) 메소드는 계정의 생성이나 비밀번호 변경을 시도하는 사용자의 id와 password를 인자로 받아 비밀번호 적합성 검사를 수행하고 기준에 적합하면 true를, 아니면 false를 리턴해야 한다.

인터페이스를 구현한 클래스를 jar 파일 형태로 "DOMAIN_HOME/lib/application" 경로에 넣었으면 사용자는 jeusadmin의 명령어를 통해 Custom Password Validator 설정을 할 수 있다.

jeusadmin에서는 Custom Password Validator에 관련한 명령어로 **add-custom-password-validator**, **remove-custom-password-validator**, **show-custom-password-validator**를 제공한다. 명령어에 대한 정보는 JEUS Reference 안내서의 "Part II. 콘솔 명령어와 툴"을 참고한다.

```

[MASTER]domain1.adminServer>add-custom-password-validator -class MyValidator
Custom password validator [MyValidator] is added successfully.
Check the results using "show-custom-password-validator".
[MASTER]domain1.adminServer>show-custom-password-validator
=====
+-----+
| custom password validator class names |
+-----+

```

```
| MyValidator |
+-----+
=====
```

이렇게 설정한 Custom Password Validator 값은 **domain.xml**에 아래와 같은 형식으로 저장된다.

Custom Password Validator로 패스워드 적합성 검사 설정 : <domain.xml>

```
<domain>
  . . .
  <security-manager>
    . . .
    <password-validator>
      <custom-password-validator>
        <class-name>MyValidator</class-name>
        <class-name>MyValidator2</class-name>
      </custom-password-validator>
    </password-validator>
    . . .
  </security-manager>
  . . .
</domain>
```

Custom Password Validator의 설정 변경사항을 적용하기 위해서는 서버를 재시작해야 한다. 서버 재시작 이후부터 새 사용자의 패스워드를 입력하거나 기존 패스워드를 변경하려고 할 때 설정된 Custom Password Validator의 Password Validation이 적용된다. 만약 사용자가 여러 개의 클래스를 구현하여 Custom Password Validator에 등록하였다면 모든 클래스의 Password Validation을 통과해야만 비밀번호 설정을 끝마칠 수 있다.



서버를 재시작하는 과정에서 "DOMAIN_HOME/lib/application"에 사용자가 설정을 끝마친 Custom Password Validator에 등록된 클래스의 jar 파일이 없으면 서버는 부팅에 실패한다. 이 경우 사용자가 직접 **domain.xml**을 올바르게 수정해 주어야 서버를 다시 시작할 수 있다.

암호화 알고리즘

사용자 설정을 할 때는 사용자의 패스워드를 지정해야 한다. 패스워드를 plain-text 형식으로 저장하는 것이 가능하나 암호화 알고리즘을 통해 암호화하여 타인이 알아볼 수 없도록 한다.



안정성이 입증된 대칭키 알고리즘인 AES를 사용하는 것을 권장한다.

jeusadmin의 **set-password** 명령어를 통해 사용자별로 패스워드의 암호화가 가능하며, JEUS_HOME/bin/encryption을 이용하여 직접 암호화할 수도 있다.

직접 암호화한 경우는 패스워드를 설정할 때 다음의 형식으로 설정한다.

```
{알고리즘}암호화된문자열
```


대칭키 암호화 알고리즘의 키 값은 JEUS_HOME/domains/<domain name>/config/security/security.key 파일에 저장된다.

다음은 비밀번호 암호화에서 지원하는 알고리즘에 대한 설명이다.

항목	설명
AES/DES/DESeed/SEED/Blowfish	대칭키 암호화 알고리즘이다.
base64	인코딩 알고리즘이다. Base64로 인코딩된 정보는 누구나 손쉽게 디코딩할 수 있어 보안상 안전하지 않다.
SHA	Hash 알고리즘이다. 복호화가 불가능하다.

암호화 알고리즘을 사용할 때 key의 크기를 지정할 수 있다. Key 크기는 관리자가 시스템 프로퍼티로 관리하며 시스템 전체에 통일된 key 크기가 적용된다.

Key 크기의 기본값은 256bit 이며, 시스템 프로퍼티로 jeus.security.keylength 옵션을 주어 key 크기를 변경할 수 있다. 예를 들어 "-Djeus.security.keylength=256" 옵션을 주어 256bit의 key 크기를 갖는 암호화 알고리즘을 사용할 수 있다.

시스템 프로퍼티로 설정한 key 크기가 암호화 알고리즘이 지원하는 최대 key 크기 값보다 큰 경우, 암호화 알고리즘이 지원하는 최대값으로 key 크기가 설정된다. 예를 들어 AES는 key 크기로 128, 192, 256bit만 지원하기 때문에 AES512와 같은 설정은 AES256으로 적용된다.

만약 지정한 key 크기가 암호화 알고리즘이 지원하는 최대 key 크기 값보다 작으면서 지원하지 않는 key 크기인 경우에는 EncryptionException이 발생한다. 예를 들어 AES는 key 크기로 128, 192, 256bit만 지원하기 때문에 AES200과 같은 설정은 EncryptionException이 발생한다.



새롭게 시스템 프로퍼티를 설정한 이후에는 반드시 새롭게 password 초기화 과정을 거쳐야 한다.

마스터 패스워드를 통한 SecretKey 파일 관리

암호화 툴(JEUS_HOME/bin에 위치)을 사용해서 패스워드를 암호화하는 경우 해당 암호화 파일에 적용되는 SecretKey 정보를 security.key 파일에 암호화 알고리즘별로 저장하여 관리한다. 마스터 패스워드를 입력받아 이 패스워드로 security.key 파일을 암호화하여 저장할 수 있다.

security.key 파일은 다음 경로에 위치한다. 도메인 환경을 구성하는 경우 해당 security.key 파일을 함께 다른 노드에 옮겨놓아야 한다.

```
JEUS_HOME/domains/<domain name>/config/security/security.key
```

JEUS에 등록하는 데이터베이스 패스워드를 암호화하였을 경우 클라이언트에서는 이를 decrypt하기 위해 Key가 필요하다. 이때 SecretKey 파일 경로를 시스템 프로퍼티를 이용해 설정할 수 있다.

또한, security.key 파일에 마스터 패스워드가 설정된 경우 마스터 패스워드도 시스템 프로퍼티를 이용해서 설정할

수 있다. key path를 지정하는 프로퍼티 이름은 jeus.security.keypath이며, 마스터 패스워드를 설정하려면 jeus.security.master를 이용하면 된다. 이 프로퍼티들은 JEUS를 기동할 때도 이용할 수 있다. 다만, 보안의 이유로 마스터 패스워드는 프롬프트(standard input)로 입력하는 것을 권장한다.

2.5.4. 로그인 정보 캐시 기능

JEUS 스크립트를 통한 서버 부트나 jeusadmin으로 서버에 접속할 때는 사용자 정보가 필수로 입력되어야 한다. 이때 사용자 정보를 매번 입력하지 않고 로그인 정보를 캐시하여 사용할 수 있는 기능을 제공한다.



해당 정보는 USER_HOME/.jeusadmin/.jeuspasswd 파일에 AES로 인코딩되어 저장된다. 캐시 로그인 기능은 JEUS_HOME/bin/security.key 파일을 통해 인코딩/디코딩을 수행한다. 파일에 사용자의 중요한 정보인 id/password가 저장되기 때문에 본 기능의 사용은 권장하지 않는다.

로그인 정보의 Cache는 한 번이라도 인증이 이루어진 다음 파일에 기록된다.

JEUS 스크립트를 통한 서버 부트나 jeusadmin의 **connect** 명령어를 수행할 때 사용자 정보와 함께 **-cachelogin** 옵션을 추가하면 해당 로그인 정보가 파일에 저장된다. 이때 캐시 파일에 **<domain name>:<user name>**을 Key로 로그인 정보가 저장된다.

추후 JEUS 스크립트가 실행될 때 사용자 패스워드를 입력하지 않아도 도메인 이름과 사용자 이름에 대한 로그인 정보가 존재하면 자동으로 사용자 정보를 채워준다. 캐시된 로그인 정보가 존재한다 하더라도 사용자가 직접 사용자 정보를 입력한 경우에는 캐시된 로그인 정보는 무시된다.

다음 예제는 저장된 로그인 정보를 보여준다.

```
#Warning: We don't recommend to use this on Production Environment.
domain1:user1
gEPaqBz6BaXWxdSXf8wZNPLsWkysgcov/KJnHvDeduKRVTAOb7F6zRaPhc2zLBUIUi46FQFWn14mQiEIUbG9UEe4yZrsRri7yS9qi+7EwA=
```

저장된 로그인 정보는 jeusadmin의 off-line 명령어인 **remove-login-cache**를 통해 삭제할 수 있다.

2.6. 보안 시스템 정책 설정

디폴트 보안을 설정하는 경우 정책 데이터(권한 부여 데이터)는 **policies.xml** 파일에서 읽어온다.

이 파일은 다음 경로에 위치한다.

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/policies.xml
```

<domain name>은 도메인의 이름이고, **<security domain name>**은 정책이 적용될 보안 도메인의 이름을 나타낸다.

2.6.1. XML 편집 설정

policies.xml의 XML 스키마는 **policies.xsd**로 JEUS_HOME/lib/schemas/jeus/supportLocale/ko 디렉터리 내에 있다. policies.xml은 0개 이상의 **<policy>** 태그로 구성되어 있으며, 각 태그는 개별 정책(권한 부여 데이터)을 나타낸다.

보안 시스템 Policy 설정 : <policies.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <role-permissions>
      <role-permission>
        <principal>user1</principal>
        <role>administrator</role>
      </role-permission>
      <role-permission>
        <principal>user1</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>jndi</resource>
        <actions>lookup</actions>
        <unchecked/>
      </resource-permission>
      <resource-permission>
        <role>administrator</role>
        <resource>jeus.*</resource>
        <actions>*</actions>
      </resource-permission>
    </resource-permissions>
  </policy>
</policies>
```

다음은 설정 태그에 대한 설명이다.

- <role-permissions> (0 또는 1)

Principal-to-Role 매핑들에 대한 정보를 제공한다. 하위 태그로 0개 이상의 <role-permission> 태그를 가지고

있다.

<role-permission> 태그는 다음의 태그로 구성되어 있다.

태그	설명
<principal> (0개 이상)	현재 <role-permission>을 소유하고 있는 Principal 이름이다.
<role> (1개, 필수)	Role 이름을 나타낸다.
<actions> (선택)	Role에 대한 액션을 나타낸다.
<classname> (선택)	Role Permission으로 사용될 java.security.Permission을 구현한 Java 클래스명이다. 생략되면 jeus.security.resource.RolePermission이 디폴트로 사용된다. Custom Permission을 설정하고 구현하는 방법은 " Custom Permission 구현 및 설정 "을 참고한다.
<excluded> (선택)	값이 없는 empty 태그(<xxx/>)이다. 설정되어 있으면 해당 Role Permission이 배제된다. 즉, Permission이 암시하는 Role에 아무도 접근할 수가 없다.
<unchecked> (선택)	값이 없는 empty 태그이다. 설정되어 있으면 Role Permission이 체크되지 않는다. 즉, Permission이 암시하는 Role에 누구나 접근할 수 있다.

- <resource-permissions> (0개 이상)

Role-to-Resource에 대한 정보는 다음의 태그로 구성되어 있다.

- <context-id> (선택)

문자열로 Context ID를 나타낸다. Context는 권한 부여 체크가 일어나는 범위를 나타낸다. JEUS 시스템이 JEUS 리소스에 대해 사용하는 기본 Context ID는 "default"이다.

- <resource-permission> (0개 이상)

다음의 항목으로 구성되어 있다.

태그	설명
<role> (0개 이상)	현재 Resource Permission을 소유하고 있는 Role 이름이다.
<resource> (1개, 필수)	리소스 이름을 나타낸다.
<actions> (선택)	리소스에 대한 액션을 나타낸다.
<classname> (선택)	Resource Permission으로 나타내는 java.security.Permission을 구현한 클래스명이다. 생략하면 jeus.security.resource.ResourcePermission이 디폴트로 사용된다. Custom Permission을 생성하고 설정하는 방법은 " Custom Permission 구현 및 설정 "을 참고한다.
<excluded> (선택)	값이 없는 empty 태그이다. 설정하면 해당 Resource Permission이 배제된다. 즉, Permission이 암시하는 리소스는 누구도 접근할 수 없다.
<unchecked> (선택)	값이 없는 empty 태그이다. 설정하면 해당 Resource Permission을 체크하지 않는다. 즉, Permission이 암시하는 리소스는 누구나 접근할 수 있다.

보통 policies.xml은 JEUS 서버 리소스에 대한 Permission(JNDI, JMS, Security 서버 등)을 설정하는 데 사용되고, Jakarta EE 애플리케이션과 Jakarta EE 모듈에 대한 Permission을 설정하는 데는 사용하지 않는다. 대신 Jakarta EE 애플리케이션과 모듈에 Permission을 설정하기 위해 다양한 DD 파일을 사용한다. 자세한 사항은 [애플리케이션과 모듈에서 보안 설정](#)을 참고한다.

Custom Permission 구현 및 설정

Permission(Role Permission 또는 Resource Permission)을 policies.xml 파일에 추가할 때마다 Permission을 나타내는 Java 클래스명을 설정해야 한다. 해당 클래스는 java.security.Permission 추상 클래스를 확장한 것이다. 자신만의 java.security.Permission 구현 클래스를 생성하여, policies.xml에 <classname> 태그로 설정하면 Custom Permission을 직접 생성할 수 있다.

다음 요구사항을 가진 새로운 Custom Permission을 구현하는 방법을 설명한다. 새로운 Role Permission은 원래의 Role 외에 다음과 같은 2가지 조건이 충족되는 상황에서 또 다른 Role을 암시할 수 있다.

- 또 다른 Role도 원래의 Role과 동일한 이름을 가지고 있다.
- 현재 시간이 특정 시간 범위(오전 9시부터 오후 5시까지) 내에 있을 때만 또 다른 Role을 암시할 수 있다.

예를 들어 이러한 Role Permission은 बैंकिंग 애플리케이션에서 "user2"라는 Principal이 오전 9시부터 오후 5시까지 업무시간 동안만 "teller"라는 Role을 가질 수 있도록 만들 수 있다. 이를 위해 다음의 2가지 작업이 필요하다.

1. Custom Permission 클래스를 구현한다. 클래스 구현이 완료되면 "javac"로 컴파일하고 JEUS 서버의 클래스 패스에 해당 클래스 경로를 설정한다. 다음은 위의 요구사항을 만족하는 Custom Permission 클래스를 구현한 것이다. 자세한 코드는 생략한다.

Custom Permission 클래스 : <TimeConstrainedRolePermission.java>

```
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;
        }
    }
}
}
}

```

해당 클래스는 `jeus.security.resource.RolePermission`을 상속하고, `jeus.security.resource.RolePermission`은 다시 `java.security.Permission`을 상속한다. 이는 코드 재사용성을 높이기 위한 구조이다.

`TimeConstrainedRolePermission`을 상속해 또 다른 `java.security.Permission`을 만드는 것도 가능하다.

위의 소스 일부에서 보면 2가지 타입의 생성자가 있다.

- 첫 번째는(role name) 하나의 파라미터만 받는다.
- 두 번째는(role name, time constraint) 2개의 파라미터를 받는다.

`java.security.Permission`에서 첫 번째 파라미터는 "name"을 뜻하고 두 번째 파라미터는 "actions"를 뜻한다. 몇몇 `Permission` 구현 클래스는 "actions" 파라미터를 받는 두 번째 타입의 생성자가 생략되기도 한다.

- "actions" 파라미터는 `Permission`에 대한 유효 시간을 나타내며, "09:00-17:00"라는 값을 가진다.

- "name" 파라미터는 "administrator" 또는 "teller"와 같은 Role명을 나타낸다.

소스의 핵심은 "implies(Permission anotherPermission)" 메소드로 현재 시스템 시간이 주어진 유효 시간 내에 있는지 체크한 다음, 만약 그렇다면 super.implies() 메소드를 호출하고, 그렇지 않으면 false를 리턴한다. 모든 implies() 메소드는 Boolean 값을 리턴하게 되어 있는데, 해당 Permission이 파라미터로 넘어온 Permission을 암시하는지를 나타낸다.



1. implies() 메소드와 java.security.Permission 추상 클래스에 대한 자세한 정보는 Java SE JavaDoc 문서를 참고한다.
2. jeus.security.resource.RolePermission, jeus.security.resource.TimeConstrainedRolePermission, jeus.security.resource.ResourcePermission에 대한 자세한 정보는 [참고 자료](#)와 JEUS Security JavaDoc에서 jeus.security.resource 패키지를 참고한다.

2. policies.xml에 해당 Permission을 사용하도록 설정한다.

Custom Permission 클래스 : <policies.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <role-permissions>
      <role-permission>
        <principal>user2</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>
```

"user2"라는 사용자는 "administrator" Role을 오전 9시부터 오후 5시까지 업무시간 동안만 부여받는다. "administrator" Role은 모든 JEUS 리소스(jeus.*)에 대해 모든 액션(*)을 실행할 수 있는 권한을 가지고 있다. 따라서, user2는 업무시간 동안만 모든 JEUS 리소스에 대한 모든 권한을 행사할 수 있다.

위의 스키마는 Jakarta EE 애플리케이션과 모듈에서 JEUS DD 파일을 설정할 때도 그대로 적용되고 예제에서 보여준 Principal-to-Role Permission에 적용되고, Role-to-Resource Permission에도 동일하게 적용된다. JEUS DD에서 Custom Permission을 사용하는 방법은 [애플리케이션과 모듈에서 보안 설정](#)을 참고한다.

2.6.2. 데이터베이스 사용 설정

데이터베이스를 이용하여 Policy를 설정하려면 다음과 같이 **security-domains.xml**에서 보안 도메인 서비스를 지정해야 한다. Authentication과 마찬가지로 Driver Manager를 직접 이용하고 싶은 경우 <datasource-id> 대신에 <dbdriver-config> 태그를 추가한다. 그러나 일반적으로 JEUS JDBC 데이터소스를 사용할 것을 권장한다.

데이터베이스를 이용한 Policy 설정 : <security-domains.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<security-domains xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9.0">
  ...
  <security-domain>
    <name>MY_DOMAIN</name>
    <authorization>
      <repository-service>
        <database-repository>
          <datasource-id>auth</datasource-id>
        </database-repository>
      </repository-service>
    </authorization>
    . . .
  </security-domain>
  . . .
</security-domains>
```

데이터베이스를 사용하는 AuthenticationRepositoryService를 사용하려면 데이터베이스에 접근하는 JDBC 드라이버와 접근을 하기위한 정보인 URL, 사용자 이름 그리고 비밀번호가 필요하다.

본 예제에서는 Oracle에 접근하기 위한 정보로 비밀번호는 base64로 인코딩된 문자가 입력된다. 특정 암호화 알고리즘 또는 인코딩 방식이 적용된 비밀번호 값을 기입하는 경우 accounts.xml의 사용자 비밀번호와 동일한 방식으로 기입하면 된다.

데이터베이스에서 사용되는 테이블은 다음과 같이 구성된다.

jeus_role_principal		
PK	domain	VARCHAR(30)
PK	rolepermissionid	VARCHAR(30)
PK	username	VARCHAR(30)

jeus_role_permissions		
PK	domain	VARCHAR(30)
PK	rolepermissionid	VARCHAR(30)
	description	VARCHAR(100)

jeus_role_permission		
PK	rolepermissionid	VARCHAR(30)
PK	role	VARCHAR(30)
	actions	VARCHAR(100)
	classname	VARCHAR(100)
	excluded	VARCHAR(5)
	unchecked	VARCHAR(5)

jeus_seqno		
PK	tablename	VARCHAR(30)
PK	currentseqno	VARCHAR(30)

jeus_resource_permissions		
PK	domain	VARCHAR(30)
PK	contextid	VARCHAR(30)
PK	resourcepermissionid	VARCHAR(30)
	description	VARCHAR(100)

jeus_resource_permission		
PK	domain	VARCHAR(30)
PK	res	VARCHAR(30)
	actions	VARCHAR(100)
	classname	VARCHAR(100)
	excluded	VARCHAR(5)
	unchecked	VARCHAR(5)

jeus_resource_role		
PK	domain	VARCHAR(30)
PK	rolepermissionid	VARCHAR(30)
PK	resourcepermissionid	VARCHAR(30)

Policy를 저장하기 위한 데이터베이스 테이블 구조

데이터베이스에 테이블이 존재하지 않는다면 JEUS_HOME/templates/security/dbrealm.sql.template 파일을 이용해서 DB에 테이블을 생성하여 기본적인 Policy는 “SYSTEM_DOMAIN”에 “administrator”라는 Role을 가지게 되고, “jeus.*”에 대한 Resource 권한을 가진다. “administrator” Role 포함된 principal은 [데이터베이스 사용 설정](#)에서 설명한 jeus이다.

2.7. 추가 항목 설정

본 절에서는 Subject와 Policy 이외의 추가 항목을 설정하는 방법을 설명한다.

2.7.1. Java SE SecurityManager 설정

JEUS에서 Java SE SecurityManger를 사용하면, 플랫폼에 대한 부수적인 견고성을 얻을 수 있으나 성능이 저하된다. 일반적으로 Java SE SecurityManger에서는 모든 핵심 JEUS 코드뿐 아니라 JEUS에 deploy되어 있는 Jakarta EE 애플리케이션 및 모듈이 완벽하게 신뢰받은 코드로 간주되므로, 굳이 보안 관리자를 사용하여 부하를 초래할 필요는 없다. 보안 관리자를 사용하지 않는 모드가 JEUS에서 디폴트 모드이다.

그러나 때때로 Java SE SecurityManger를 사용하여 코드 수준의 보안을 강화시켜 추가적인 견고성을 높이는 문제가 성능 저하보다 중요하게 다루어지는 경우가 있다.

예를 들어 시스템 관리자가 불안정한 코드를 포함하고 있을지도 모르는 Jakarta EE 애플리케이션을 JEUS에 deploy해야 한다. 이 경우에 성능 저하라는 비용을 감수하고서라도 코드 레벨의 보안을 강화시켜 호스트를 보호하는 것이 더 중요하다고 판단되면 Java SE SecurityManger를 작동시켜야 한다.

Java SE SecurityManger를 JEUS와 함께 동작시키기 위해 domain.xml에 특정 서버에 대해 jvm-option을 다음과 같이 정의한다.

```
-Djava.security.manager  
-Djava.security.policy=${JEUS_HOME}/domains/domain1/config/security/policy(UNIX기준)
```

정책 파일은 JEUS_HOME/domains/<domain name>/config/security/policy이며 내용은 다음과 같다.

Java SE SecurityManager 설정 : <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 java.security.SecurityPermission "putProviderProperty.*";  
    permission java.security.SecurityPermission "insertProvider.*";  
    permission javax.management.MBeanPermission "*", "*";  
};
```

Java SE SecurityManger는 JEUS의 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 Permission이 있는가)를 다루고 있다.

둘 사이 유일한 접점은 JEUS가 특별한 경우 Java SE SecurityManger를 사용하여 코드 수준의 Permission을 체크함으로써 악의적인 Servlet이나 EJB 코드로부터 JEUS를 보호하기도 한다는 점이다.

2.7.2. JACC Provider 설정

JEUS에서 JACC 2.0 설정에 대한 이슈는 [JACC Provider 사용](#)에서 설명할 것이다. 더 자세한 내용은 해당 절을 참고한다.

2.7.3. Identity 부여를 위한 정보 설정

IdentityAssertionService 지원하는 경우 추가적으로 인증서와 사용자 간의 매핑 정보가 있는 **cert-user-map.xml** 파일에서 읽어 들인다. 이 파일은 다음의 경로에 위치한다.

JEUS_HOME/domains/<domain name>/config/security/<security domain name>/

<domain name>은 도메인의 이름이고, <security domain name>은 사용자가 속한 보안 도메인의 이름을 나타낸다.

cert-user-map.xml의 XML 스키마는 cert-user-map.xsd이며, JEUS_HOME/lib/schemas/jeus/supportLocale/ko 경로에 있다.

이 파일에는 최상위 태그 **<cert-user>** 하위에 <user> 및 <cert> 태그가 있으며 하위 0개 이상의 <cert-user> 태그가 포함되어 있다. 각각은 사용자에게 대한 인증서 매핑을 위한 속성 정보를 나타내고 있다.

Identity 부여를 위한 정보 설정 : <cert-user-map.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cert-user-map xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <cert-user>
    <username>user1</username>
    <cert>
      <subjectDN>user1DN</subjectDN>
    </cert>
  </cert-user>
</cert-user-map>
```

각 <cert-user> 태그는 다음과 같은 하위 태그를 가지고 있다.

- <username> (필수)

인증서 attribute 값에 매핑되는 사용자에게 대한 이름을 나타낸다. (예: 사용자명, 사용자 ID)

- <cert>

각 <cert> 태그는 다음과 같은 하위 태그를 가지고 있으며 사용자에게 대한 인증서 매핑 정보를 정의한다. 이는 Truststore 파일에 포함된 인증서에 대해서 하위 attribute 값들이 유일한 ID를 보장해서 identity를 부여할 수 있도록 사용자 이름과 매핑되는 값들을 정의한다.

태그	설명
<alias> (선택)	Keystore 내의 Certificate에 대한 Alias를 정의한다. (문자열)
<subjectDN> (선택)	Keystore 내의 Certificate에 대한 subjectDN를 정의한다. (문자열)
<SKI> (선택)	Keystore 내의 Certificate에 대한 SKI를 정의한다. (문자열)
<issuer> (선택)	Keystore 내의 Certificate에 대한 issuer를 정의한다. (문자열)
<serialNo> (선택)	Keystore 내의 Certificate에 대한 serial number를 정의한다. (문자열)

2.7.4. Identity에 대한 인증서 정보 설정

JEUS 보안 시스템에서는 인증서로 인증된 Identity 사용자 정보에 대한 인증서 정보를 얻을 수 있는 API를 제공하는 UserCertMappingService를 제공한다. UserCertMappingService 지원하는 경우 추가적으로 사용자별 인증서

파일을 얻기 위한 매핑 정보가 있는 **user-cert-map.xml** 파일에서 읽어 들인다. 이 파일은 다음의 경로에 위치한다.

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/
```

<domain name>은 도메인의 이름이고, <security domain name>은 사용자가 속한 보안 도메인의 이름을 나타낸다.

user-cert-map.xml의 XML 스키마는 user-cert-map.xsd이며, JEUS_HOME/lib/schmas/jeus/supportLocale/ko 경로에 있다.

이 파일에는 최상위 태그 **<user-cert-map>** 태그 하위에 0개 이상의 <user-cert> 태그가 포함되어 있다. 각각은 Keystore 파일에서 사용자에게 대한 인증서를 얻기 위한 속성 정보를 나타낸다.

Identity에 대한 인증서 정보 설정 : <user-cert-map.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user-cert-map xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <user-cert>
    <username>user1</username>
    <alias>alias1</alias>
    <keypassword>changeit</keypassword>
    <secretkey>
      <keyname>testkeypass</keyname>
      <keyalgorithm>AES</keyalgorithm>
      <keyvalue>bjhTUjJvSXRTOGlkVEd1NHJnM2N3Vn1jSDZXV0JkYz0=</keyvalue>
    </secretkey>
  </user-cert>
</user-cert-map>
```

각 <user-cert> 태그는 다음과 같은 하위 태그를 가지고 있다.

- <username> (필수)

Keystore 내의 Certificate에 대한 사용자 이름을 정의한다. "primary" identify로서 유일해야 한다.

(예: 사용자명, 사용자 ID)

- <alias> (필수)

Keystore 내의 Certificate에 대한 Alias를 정의한다.

- <keypassword> (선택)

Keystore 내의 Certificate에 대한 Private Key를 얻기위한 Keypassword를 정의한다. (예: changeit)

- <secretkey> (선택)

Private Key를 정의한다.

각 <secretkey> 태그는 다음과 같은 하위 태그를 가지고 있다.

태그	설명
<keyname> (필수)	Private Key의 이름을 정의한다. <user-cert-map> 태그 하위에 존재하는 keyname 값은 유일해야 한다. (문자열)
<keyalgorithm> (필수)	Private Key의 키 알고리즘을 나타낸다. (문자열)
<keyvalue> (필수)	Private Key의 값을 Base64 형태로 나타낸다.

3. 애플리케이션과 모듈에서 보안 설정

본 장에서는 Jakarta EE 애플리케이션과 EJB 모듈, 웹 모듈에서 보안을 설정하는 방법을 설명한다.

3.1. 개요

본 절에서는 애플리케이션과 모듈에 보안을 설정하는 것과 관련된 가장 기본적인 이슈를 설명한다.

Jakarta EE 애플리케이션, EJB 모듈, 웹 모듈의 가장 기본적인 보안 설정 과정은 다음과 같다.

1. Role-to-Resource 매핑을 각 모듈마다 설정한다.
2. 각 애플리케이션에 대해 Principal-to-Role 매핑을 정의한다.

3.1.1. 모듈 Deployment 대 애플리케이션 Deployment

JEUS에서 deploy하는 방법은 크게 2가지로 나뉘어진다.

- EJB 모듈(.jar 파일), 웹 모듈(.war 파일)을 각각 독립적으로 deploy하는 방법
- 여러 개의 EJB 모듈, 웹 모듈, Connector 모듈(.rar)을 EAR로 압축하여 Jakarta EE 애플리케이션(.ear)으로 deploy하는 방법

3.1.2. Role-to-Resource 매핑

Assembler는 EJB 모듈이나 웹 모듈을 deploy하기 전에 해당 모듈에 Role-to-Resource 매핑을 설정해야 한다. Role-to-Resource 매핑을 보안 제약(security constraint)이라고 하며, 논리적인 Role에 모듈의 리소스를 매핑하는 것을 의미한다.

- EJB 모듈에서 리소스는 EJB의 메소드를 의미하며, META-INF/ejb-jar.xml에 설정한다.
- 웹 모듈에서 리소스는 서블릿 URL을 의미하며, WEB-INF/web.xml에 설정한다.

다음은 EJB 모듈에 설정된 보안 제약 내용으로 META-INF/ejb-jar.xml에 설정에 대한 설명이다.

EJB 모듈에 설정된 보안 제약 : <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar xmlns="https://jakarta.ee/xml/ns/jakartaee">
  <display-name>product</display-name>
  <enterprise-beans>
    <entity>
      <ejb-name>product</ejb-name> ①
      . . .
      <security-role-ref> ②
        <role-name>cust</role-name>
        <role-link>customer</role-link>
      </security-role-ref>
    </entity>
  </enterprise-beans>
```

```

<assembly-descriptor>
    . . .
    <security-role> ③
        <role-name>administrator</role-name>
    </security-role>
    <security-role> ④
        <role-name>customer</role-name>
    </security-role>
    <method-permission> ⑤
        <role-name>administrator</role-name>
        <method>
            <ejb-name>product</ejb-name>
            <method-interfaces>Remote</method-interfaces>
            <method-name>getSecretKey</method-name>
            <method-params>
                <method-param>java.lang.Integer</method-param>
            </method-params>
        </method>
    </method-permission>
    <method-permission> ⑥
        <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> ⑦
        <role-name>customer</role-name>
        <method>
            <ejb-name>product</ejb-name>
            <method-name>test1</method-name>
        </method>
    </method-permission>
    <exclude-list> ⑧
        <method>
            <ejb-name>product</ejb-name>
            <method-interfaces>Remote</method-interfaces>
            <method-name>getCustomerProfile</method-name>
            <method-params></method-params>
        </method>
    </exclude-list>
</assembly-descriptor>
</ejb-jar>

```

DD 파일의 각 부분이 의미하는 보안 제약 사항은 다음과 같다.

- ① "product"라 불리는 Entity EJB가 있다.
- ② Role-to-Role Reference 매핑이 선언되어 있다. 실제 선언된 Role인 "customer"는 "cust"라는 Role Reference로 참조될 수 있다. 이는 EJB 소스 내에 "cust"라는 이름의 Role은 실제로는 "customer" Role에 해당한다는 것을 의미한다.
- ③ 논리적 Role인 "administrator"가 선언되어 있다.
- ④ 논리적 Role인 "customer"가 선언되어 있다.

- ⑤ Role-to-Resource 매핑이 선언되어 있는데, 이는 "administrator" Role이 리모트 EJB 인터페이스의 getSecreteKey 메소드를 호출할 수 있도록 허락한다. 이 매핑에 따라 오직 "administrator" Role에 속하는 Principal만이 getSecreteKey 메소드를 호출할 수 있다.
- ⑥ doSomeAdmin 메소드는 Unchecked로 선언되어 있다. 즉, 누구나 Role에 상관없이 이 메소드를 호출할 수 있다는 의미이다.
- ⑦ "customer" Role에 속하는 Principal은 "product" EJB의 "test1" 메소드를 호출할 수 있다.
- ⑧ getCustomerProfile 메소드는 Excluded 목록에 포함되어 있다. 즉, Role에 상관없이 어떤 Role도 해당 메소드를 호출할 수 없다는 의미이다.

웹 모듈에 대해서도 EJB와 비슷한 설정이 적용된다. 단, EJB에서 Role이 EJB 메소드를 호출할 수 있느냐의 문제였다면 웹 모듈에서는 Role이 서블릿 URL에 접근할 수 있느냐의 문제를 다룬다. 웹 모듈 보안 설정에 대한 자세한 정보는 [웹 모듈 보안 설정](#)을 참고한다.

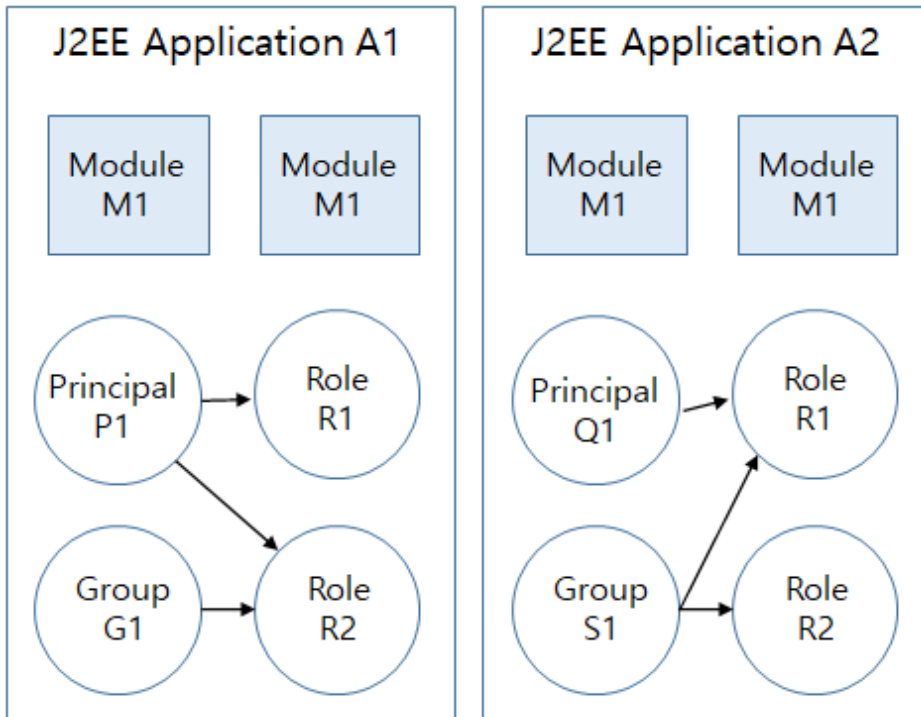
지금까지 언급한 2가지 Role, 즉 "administrator"와 "customer"는 논리적 개념으로 실제 deploy되는 환경을 의미하지 않는다. 따라서, 논리적 Role을 실제 환경의 사용자나 사용자 그룹에 매핑하는 작업이 필요하다. 이 작업을 Principal-to-Role 매핑이라 하는데 자세한 내용은 [Principal-to-Role 매핑](#)에서 설명한다.

3.1.3. Principal-to-Role 매핑

일반적으로 Deployer가 Jakarta EE 모듈 또는 애플리케이션에 정의된 Role을 실제 환경의 사용자나 사용자 그룹에 매핑하는 작업을 한다. 여기서, Principal을 논리적 Role에 매핑하는 것은 애플리케이션 범위(scope)를 가진다는 점에 주의해야 한다.

예를 들어 모듈 M1과 M2가 A1이라는 애플리케이션에 포함되어 있다면, Principal-to-Role 매핑은 M1과 M2에서 공유된다. 더 나아가 A2라는 별도의 애플리케이션은 A1과는 완전히 다른 Principal-to-Role 매핑을 가진다. A1과 A2에서 동일한 이름의 Role이 있다고 하더라도 이들은 서로 다른 Role로 인식되며 공유되지 않는다.

다음 그림은 이러한 범위와 관련된 개념에 대한 설명이다.



Principal-to-Role 매핑

위 그림에서 보듯이 2개의 서로 다른 애플리케이션(A1, A2)가 서로 다른 Principal-to-Role 매핑을 가지고 있으며, 포함되어 있는 모듈(M1, M2)은 애플리케이션이 포함하고 있는 매핑을 공유한다.

JEUS에서 Principal-to-Role 매핑은 다음의 파일에 설정된다.

- JEUS DD인 jeus-ejb-dd.xml(EJB 모듈)
- jeus-web-dd.xml(웹 모듈)
- jeus-application-dd.xml(Jakarta EE 애플리케이션)

다음은 ejb-jar.xml에 대한 jeus-ejb-dd.xml의 설정 예이다.

Principal-to-Role 매핑 : <jeus-ejb-dd.xml>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    <role-permission> ①
      <principal>user1</principal>
      <role>administrator</role>
      <classname>mypackage.MyRolePermission</classname>
    </role-permission>
    <role-permission> ②
      <principal>user2</principal>
      <role>customer</role>
    </role-permission>
    <role-permission> ③
      <role>roleA</role>
      </excluded>
    </role-permission>
    <role-permission> ④
      <role>roleB</role>
      </unchecked>
    </role-permission>
  </module-info>
</jeus-ejb-dd>
```

```

<unspecified-method-permission> ⑤
    <role>administrator</role>
</unspecified-method-permission>
</module-info>
<beanlist>
    . . .
</beanlist>
</jeus-ejb-dd>

```

위의 예제는 다음과 같은 보안 정보를 담고 있다.

- ① "user1"이라는 Principal(user)에게 "administrator" Role을 허가하는 Role Permission이 선언되어 있다. 이는 "user1"이 "administrator" Role의 모든 권한을 가지게 되었다는 것을 의미한다.
예제에서 보듯이 Custom Role Permission은 <classname> 태그로 정의한다. Custom Role Permission에 대한 자세한 정보는 [보안 시스템 설정](#)을 참고한다.
- ② "user2"라는 Principal(user)에게 "customer" Role을 부여하는 Role-Permission이 선언되어 있다. 이는 "user2"가 "customer" Role이 가지는 모든 권한을 가지게 되었다는 의미이다. <classname> 태그가 설정되어 있지 않았으므로, 디폴트 Role Permission이 설정되어 있다.
- ③ "roleA"라는 Role은 모든 Principal로부터 배제되어(excluded) 있다. 즉, 어떤 Principal도 "roleA"에 포함될 수 없다.
- ④ "roleB"라는 Role은 권한 부여 시스템에서 체크되지 않는다.(unchecked) 즉, 모든 Principal은 자동으로 "roleB"에 포함된다.
- ⑤ <unspecified-method-permission>은 "unspecified" EJB 메소드를 기본적으로 어떻게 다룰지 결정한다. "unspecified" EJB 메소드는 ejb-jar.xml에서 <method-permission> 태그로 언급되지 않은 메소드를 의미한다. <unspecified-method-permission>은 3가지 방식으로 처리할 수 있다. Role에 매핑(예제에서는 "administrator" Role에 매핑되었다)하거나, excluded(누구도 접근할 수 없는 것)로 취급하거나, unchecked(누구나 접근할 수 있는 것)로 취급하는 것이 그것이다. 마지막 unchecked 옵션이 기본값이다.



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

3.1.4. 사용자 설정

사용자 설정은 크게 2가지로 나누어진다.

- 애플리케이션에서 사용하는 도메인 자체의 사용자 설정을 이용하는 방식으로 다음의 파일에 설정한다. ([보안 시스템 사용자 정보 설정](#) 참조)

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/accounts.xml
```

- EJB 모듈(.jar 파일), 웹 모듈(.war 파일), Jakarta EE 애플리케이션(.ear) 자체의 사용자 설정 방식으로 WEB-INF(웹 모듈) 또는 META-INF(EJB 모듈 및 EAR) 디렉터리 안의 **accounts.xml**을 설정한다.



사용자 정보는 위의 어떤 방식으로 설정을 했는지에 관계없이 애플리케이션의 보안 도메인 범위(scope)을 가진다. 즉, 한 보안 도메인에서의 사용자는 같은 보안 도메인을 사용하는 모든 애플리케이션 내에서 공유된다.

3.2. EJB 모듈 보안 설정

본 절에서는 EJB 모듈에 보안 설정하는 방법을 자세하게 설명한다. 본 절에서 다루지 않은 CSI와 같은 보안 설정들은 "JEUS EJB 안내서"를 참고한다.

보안 설정 방법은 크게 2가지로 나눌 수 있다.

- [ejb-jar.xml 설정](#)
- [jeus-ejb-dd.xml 설정](#)

3.2.1. ejb-jar.xml 설정

ejb-jar.xml에 다음과 같은 보안 요소를 설정할 수 있다.

- Role-to-Role Reference 매핑
- security identity 설정
- 논리적 Role 선언
- EJB 메소드 Permission 설정

다음 예제는 위 보안 설정 요소를 포함하는 ejb-jar.xml이다.

보안 설정 : <ejb-jar.xml>

```
<?xml version="1.0"?>
<ejb-jar xmlns="https://jakarta.ee/xml/ns/jakartaee">
  <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>

```

ejb-jar.xml에 다음과 같은 보안 요소를 설정할 수 있다.

• Role-to-Role Reference 매핑

Role-to-Role Reference 매핑은 <security-role-ref> 태그로 정의한다. 목적은 실제 논리적 Role에 대한 참조를 설정하는 것이다. 그래서 EJB 소스에서는 실제 Role 대신 Role Reference를 사용하게 된다.

다음은 <security-role-ref>의 하위 태그에 대한 설명이다.

태그	설명
<role-name>	실제 Role에 대한 참조명으로 EJB 코드에는 실제 Role 이름 대신 참조명을 사용한다.

태그	설명
<role-link>	<security-role> 태그로 선언한 실제 Role 이름으로 Role Reference는 <role-link>가 가리키는 Role에 매핑된다.

• Security Identity 설정

모듈 내에 있는 각 EJB는 다른 EJB로 원격지를 호출할 때 자신의 Security Identity를 전파한다. EJB의 Security Identity는 <ejb-jar> <enterprise-beans> <type> <security-identity> 태그로 설정한다.

다음의 하위 태그를 설정한다.

태그	설명
<use-caller-identity>	empty 태그로 만일 설정되어 있으면, EJB를 호출할 때 호출자 자신이 Security Identity로 사용된다.
<run-as>	EJB가 호출할 때 <role-name>에 설정된 Role에 해당하는 principal을 Security Identity로 이용한다. jeus-ejb-dd.xml에 <run-as-identity> 태그가 있을 경우 이 설정은 무시된다.

• 논리적 Role 선언

<assembly-descriptor>의 하위 태그인 <security-role>는 해당 EJB 모듈에 적용되는 논리적 Role을 선언하는 태그이다. ejb-jar.xml에 언급되어 있는 모든 Role 이름은 이 태그 내에 선언되어 있어야 한다.

• EJB 메소드 Permission 설정

<assembly-descriptor>의 하위 태그인 <method-permission>는 각 EJB 메소드에 보안 제약을 설정하는 태그이다.

다음은 <method-permission>의 하위 태그에 대한 설명이다.

태그	설명
<role-name> (선택)	<security-role> 태그에 선언되어 있는 논리적 Role 이름으로 아래에 정의된 메소드에 접근할 수 있다.
<unchecked> (선택)	값이 없는 empty 태그로 설정되어 있으면 아래에 정의된 메소드는 권한 체크를 하지 않는다. 즉, Role에 상관없이 누구나 메소드에 접근할 수 있다.

태그	설명
<method> (1개 이상)	<p><role-name> 또는 <unchecked> 태그가 적용되는 메소드로 하위 태그는 다음과 같다.</p> <ul style="list-style-type: none"> • <ejb-name> : 메소드를 포함하고 있는 EJB 이름이다. • <method-intf> : 메소드를 포함하고 있는 EJB 인터페이스 타입으로 생략되면, 로컬, 리모트 인터페이스 모두를 뜻하게 된다. • <method-name> : EJB 메소드 이름이다. • <method-params> (선택) : 0개 이상의 <method-param> 태그를 가질 수 있다. <method-param> 태그는 EJB 메소드의 각 파라미터를 나타낸다. 태그가 생략된 경우 메소드 이름만 동일하다면 파라미터와 상관없이 동일한 Permission이 적용된다. EJB에는 파라미터는 다르고 메소드 이름은 동일한 여러 개의 메소드가 있을 수 있다. 이 메소드들에 공통적으로 해당 Method Permission이 적용된다는 의미이다. <method-param> 태그 값은 패키지 이름을 포함한 완전한 Java 클래스 이름이거나, int와 같은 Java Primitive 타입이 될 수 있다. <method-params>를 empty 태그로 설정하면(<method-params/>) 매개변수가 하나도 없다는 것을 의미한다.
<exclude-list> (1개)	<p>단 하나의 <exclude-list>가 올 수 있으며, Excluded되는 EJB 메소드를 정의한다. 즉, 누구도 접근할 수 없는 메소드이다. 하위 태그로 1개 이상의 <method> 태그를 가질 수 있다.</p>



예제에 대한 설명은 [Role-to-Resource 매핑](#)을 참고한다. 각 태그에 대한 자세한 설명은 EJB 스펙을 참고한다. 또한 JEUS에서 EJB를 Deploy하는 방법은 "JEUS EJB 안내서"를 참고한다.

3.2.2. jeus-ejb-dd.xml 설정

jeus-ejb-dd.xml에 다음과 같은 보안 요소를 설정할 수 있다.

- 보안 관련된 설정
- ejb-jar.xml에 설정되어 있지 않는 EJB 메소드 설정
- Run-as Identity를 사용하는 EJB Principal 설정

EJB에서 대부분 Role-to-Method 매핑은 ejb-jar.xml에 설정되어 있으나, ejb-jar.xml에서 설정되어 있지 않는 메소드를 어떻게 다룰지와 실제 Principal-to-Role 매핑을 어디서 선언할지는 여전히 해결해야 할 문제이다. 이는 EJB.jar 파일의 META-INF 디렉터리에 존재하는 jeus-ejb-dd.xml 파일에서 설정한다.

다음 예제는 위 보안 설정 요소를 포함하는 jeus-ejb-dd.xml이다.

보안 설정 : <jeus-ejb-dd.xml>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    <role-permission>
      <principal>user1</principal>
      <role>administrator</role>
      <classname>mypackage.MyRolePermission</classname>
    </role-permission>
    <role-permission>
```

```

        <principal>user2</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>user1</principal-name>
        </run-as-identity>
    </jeus-bean>
    . . .
</beanlist>
</jeus-ejb-dd>

```

jeus-ejb-dd.xml에 다음과 같은 보안 요소를 설정할 수 있다.

• 보안 관련된 설정

jeus-ejb-dd.xml에서 대부분 보안 관련된 설정의 <jeus-ejb-dd> 루트 태그 바로 아래에 있는 <module-info> 태그 내에 설정된다.

- <role-permission> (0개 이상)

다음은 <role-permission>의 하위 태그에 대한 설명이다. 각 태그는 Role(Principal-to-Role 매핑)을 정의하고 있다.

태그	설명
<principal> (0개 이상)	Role에 매핑되는 Principal 이름이다.
<role> (필수)	단 하나만 허락되며, 필수 항목으로 실제 논리적 Role 이름이다. Role Permission을 구현하는 Java 클래스의 생성자에 첫 번째 파라미터로 넘겨진다. Role 이름은 ejb-jar.xml에 선언된 Role 이름과 일치해야 한다.
<actions> (선택)	Role Permission에 대해 부가적인 정보를 제공한다. 만약 설정되어 있다면 액션 데이터는 Role Permission을 구현하는 Java 클래스의 두 번째 파라미터로 넘겨진다.

태그	설명
<classname> (선택)	Role Permission을 구현하는 클래스로 패키지 이름을 포함한 완전한 이름이어야 한다. 이 클래스는 java.security.Permission 클래스의 서브 클래스로 최소한 하나의 파라미터(Role의 이름)를 받는 public 생성자를 가지고 있어야 한다. 생략되어 있다면 jeus.security.resource.RolePermission이 사용된다.
<excluded> (선택)	empty 태그로 만약 설정되어 있다면 해당 Role은 excluded된 것으로 취급한다(누구도 Role Permission에서 언급한 Role에 접근할 수 없다). <principal>, <unchecked> 태그보다 우선순위가 높다.
<unchecked> (선택)	empty 태그로 만약 설정되어 있다면, 해당 Role은 unchecked된 것으로 취급한다(누구나 Role Permission에서 언급한 Role에 접근할 수 있다). <principal> 태그보다 우선순위가 높다.

• ejb-jar.xml에 설정되어 있지 않는 EJB 메소드 설정

ejb-jar.xml에 설정되어 있지 않는 EJB 메소드들을 설정한다. <module-info>의 <unspecified-method-permission> 태그 내에 정의된다. 설정 가능한 하위 태그는 다음과 같다.

태그	설명
<role> (선택)	설정되어 있다면 모든 설정되지 않는 메소드는 이 Role에 매핑된다.
<excluded> (선택)	설정되어 있다면 모든 설정되지 않는 메소드는 excluded 취급된다(누구도 접근할 수 없다). <role>이나 <unchecked> 태그보다 우선순위가 높다.
<unchecked> (선택)	설정되어 있다면 모든 설정되지 않는 메소드는 unchecked로 취급된다(누구나 접근할 수 있다). <role> 태그보다 우선순위가 높다.

• Run-as Identity를 사용하는 EJB Principal 설정

Run-as Identity를 사용하는 EJB에 대해 Principal을 설정은 <jeus-bean> <run-as-identity> <principal-name> 태그 내에 주어진 Principal 이름이 사용된다. Principal은 ejb-jar.xml에서 <security-identity> <run-as> <role-name> 태그 내에 설정된 Role 내에 포함되어 있어야 한다.



1. 예제에 대한 자세한 설명은 [Role-to-Resource 매핑](#)을 참고한다.
2. 각 태그에 대한 자세한 정보는 "JEUS EJB 안내서"를 참고한다.

3.3. 웹 모듈 보안 설정

본 절에서는 웹 모듈(서블릿 모듈)에 보안 설정하는 방법을 설명한다.

웹 모듈의 보안 설정은 크게 2가지로 구성된다.

- [web.xml 설정](#) : Role-to-Web Resource 매핑 설정
- [jeus-web-dd.xml 설정](#) : Principal-to-Role 매핑 설정

3.3.1. web.xml 설정

web.xml은 Web Archive(WAR 파일)에 대한 메인 DD 파일이다. 표준 Jakarta EE DD 파일로 WAR 파일의 WEB-INF 디렉터리 내에 있다.

web.xml에서 설정할 수 있는 보안 요소는 다음과 같다.

- **Run-as Identity**

웹 모듈에 포함되어 있는 서블릿은 EJB로 원격 호출을 할 때 자신의 Security identity를 전파한다.

서블릿의 Security identity는 <web-app><servlet><run-as> 태그 내에 설정되고, 다음과 같은 하위 태그를 가진다.

태그	설명
<role-name> (선택)	서블릿을 실행하는 Role로 생각하면 호출자의 identity가 사용된다.

- **Role-to-role reference 매핑**

Role-to-role reference 매핑은 <security-role-ref> 태그로 설정하며, Role Reference를 실제 논리적 Role에 매핑하는 것이 목적이다. Role Reference는 서블릿 코드에서 Role을 나타낼 때 사용된다.

web.xml에서 <security-role-ref> 태그는 <servlet> 내에 설정되며, 다음과 같은 하위 태그를 가진다.

태그	설명
<role-name>	서블릿 코드에 사용되는 Role 이름이다.
<role-link>	<security-role> 태그로 정의한 실제 Role 이름으로 <role-name> 태그에 설정된 Role Reference를 실제 Role에 매핑한다.

- **서블릿 URL 접근 Permissions (Security constraints, 보안제약)**

서블릿 URL에 접근을 제한하려는 목적으로 <web-app> 태그 바로 아래에 있는 <security-constraint> 태그를 사용한다.

<security-constraint>는 다음과 같은 하위 태그를 가진다.

- <web-resource-collection> (1개 이상)

접근 제한이 설정되는 웹 리소스의 목록을 나타낸다.

태그	설명
<web-resource-name>	웹 리소스 이름이다.
<url-patterns> (0개 이상)	<p>웹 리소스를 나타내는 URL(contextroot에 상대 경로로 지정) 패턴이다.</p> <p>다음은 URL 패턴의 예이다.</p> <ul style="list-style-type: none"> ◦ /mywebapp/* : mywebapp가 포함된 모든 URL ◦ /something : 정확한 URL ◦ *.jsp : JSP로 끝나는 모든 리소스
<http-method> (0개 이상)	<p>웹 리소스에 적용되는 HTTP 메소드를 나타낸다.</p> <ul style="list-style-type: none"> ◦ GET ◦ POST ◦ PUT

◦ <auth-constraint> (선택)

<security-constraint> 태그 내에 정의된 웹 리소스에 접근할 수 있는 Role을 설정한다. 하위 태그로는 0개 이상의 **<role-name>** 태그가 오는데 각각은 접근이 허락된 Role을 나타낸다.

Role 이름이 설정되어 있지 않고 empty 태그 (<auth-constraint/>)로 설정되어 있다면, 아무도 해당 웹 리소스에 접근할 수 없다는 의미이다(이는 웹 리소스가 excluded된다는 것과 동일하다).

<auth-constraint> 태그가 아예 생략되어 있다면 이는 웹 리소스가 unchecked된다는 것으로 Role에 상관없이 누구나 웹 리소스에 접근할 수 있다는 의미이다.

◦ <user-data-constraint> (선택)

웹 리소스에 맺어지는 커넥션에 대해 "transport guarantee"(전송 보장)할 것인가를 선언하는 태그다.

하위 태그로 <transport-guarantee> 태그를 가지며, 웹 리소스로서의 커넥션을 보장하는 레벨을 나타낸다.

설정값	설명
NONE	커넥션을 보장하지 못한다는 의미이다.
INTEGRAL	메시지의 무결성(즉, 보내진 메시지는 변경되지 않은 원본이다)을 보장하는 커넥션이라는 의미이다.
CONFIDENTIAL	메시지가 도청되는 것을 방지하기 위해 암호화되어 보내진다는 의미이다.

• 논리적 Role 선언

<security-constraint> 태그 다음에 논리적 Role을 나타내는 <security-role> 태그가 온다. 각 태그는 <role-name> 태그를 가지고 있는데, 이미 DD에 언급되었던 논리적 Role 이름을 선언하는 부분이다.

다음은 web.xml의 보안 설정 예제이다.

```

<?xml version="1.0"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee">
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    . . .
    <run-as> ①
      <role-name>R1</role-name>
    </run-as>
    <security-role-ref> ②
      <role-name>adminRef</role-name>
      <role-link>R1</role-link>
    </security-role-ref>
    . . .
  </servlet>
  <security-constraint> ③
    <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>
  <security-constraint> ④
    <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>
  <security-role> ⑤
    <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>

```

위 예제의 각각의 부분이 의미하는 내용은 다음과 같다.

- ① Run-as-Identity가 Role "R1"으로 설정되어 있는 것은 서블릿의 Security identity가 "R1"에 속한 Principal이어야 한다는 의미이다. 실제 Principal은 JEUS DD 파일에서 설정된다.
- ② <security-role-ref> 태그는 실제 Role인 "R1"을 Role Reference인 "adminRef"로 매핑해 놓았는데, 실제 서블릿 소스 내에서는 Role대신 Role Reference인 "adminRef"를 사용한다.
- ③ 첫 번째 Security constraint 설정은 "/a", "/b", "/a/*", "/b/*" URL 패턴을 가지고 있으면서 HTTP 메소드가 DELETE, PUT인 URL과 "*.asp"로 끝나는 URL은 누구도 접근할 수 없다는 것을 말하고 있다.
<auth-constraint/>로 설정되어 있기 때문이다.
- ④ 두 번째 Security constraint 설정은 "/a/*", "/b/*" 패턴을 가지고 있으면서 GET 방식인 URL과 /b/* 패턴을 가지고 있으면서, POST 방식인 URL은 오직 "R1" Role에 포함된 Principal만 접근할 수 있다는 것을 말하고 있다. 그리고 해당 커넥션은 CONFIDENTIAL을 보장한다.
- ⑤ "R1", "R2", "R3" 3개의 논리적 Role을 선언하고 있다.

웹 리소스(URL 패턴 + HTTP 메소드)가 web.xml에 별도로 설정되어 있지 않으면, 해당 리소스는 Role에 상관없이 누구나 접근할 수 있다는 의미이다. 이는 Jakarta EE에서 웹 애플리케이션 접근에 대한 기본값이다.



web.xml에 나타나는 모든 태그들에 대한 자세한 정보는 서블릿 스펙을 참고한다. 또한 JEUS에서 서블릿을 deploy하는 것에 대한 추가적인 정보는 "JEUS Web Engine 안내서"를 참고한다.

3.3.2. jeus-web-dd.xml 설정

웹 모듈에 대한 JEUS DD 파일인 jeus-web-dd.xml에서 보안을 설정하는 방법에 대해 설명한다.

• Principal-to-Role 매핑 정의

jeus-web-dd.xml에서 Principal-to-Role 매핑을 정의하기 위해 <jeus-web-dd><role-mapping> 태그를 사용한다.

다음은 <role-mapping>의 하위 태그들이다.

- <role-permission> (0개 이상)

Role에 대한 Permission(보통 Principal-to-Role 매핑)을 다음의 하위 태그에 설정한다.

태그	설명
<principal> (0개 이상)	Role에 매핑되는 Principal 이름이다.

태그	설명
<role> (1개, 필수)	논리적 Role 이름이다. Role 이름은 Role Permission을 구현하는 Java 클래스 생성자에 첫 번째 파라미터로 전달되고, web.xml에 주어진 논리적 Role 이름과 일치해야 한다.
<actions> (선택)	Role Permission에 대한 추가적인 데이터를 전달한다. 설정되어 있다면 Role Permission을 구현하는 Java 클래스 생성자의 두 번째 파라미터로 전달된다.
<classname> (선택)	Role Permission을 구현하는 Java 클래스명이다. 이 클래스는 java.security.Permission의 구현 클래스로 최소한 하나의 파라미터(String rolename)를 넘겨받는 public 생성자를 가지고 있어야 한다. 생략되면 기본적으로 "jeus.security.resource.RolePermission" 클래스가 사용된다.
<excluded> (선택)	empty 태그로 설정되어 있으면 Role은 excluded된 것으로 취급한다. 즉, 어떤 Principal도 해당 Role을 부여받을 수 없다. <principal>, <unchecked> 태그보다 우선순위가 높다.
<unchecked> (선택)	empty 태그로 설정되어 있으면 Role은 unchecked된 것으로 취급한다. 즉, Principal에 상관 없이 누구나 Role을 부여받을 수 있다. <principal> 태그보다 우선순위가 높다.

• 서블릿에서 <run-as-principal> 설정

서블릿에서 <run-as-principal>을 설정하기 위해 <servlet> 태그를 선언하고 <run-as><principal-name> 태그를 추가한다. <principal-name> 태그 값은 web.xml에서 <run-as><role-name>에 선언된 Role에 속하는 Principal이어야 한다.



현재로는 web.xml에서 언급되어 있지 않는 서블릿 URL을 어떻게 다룰지 정의하는 태그가 jeus-web-dd.xml에 없다. 서블릿 스펙에서는 모든 정의되지 않은 URL을 항상 Unchecked로 취급한다. 즉, 누구나 접근할 수 있다.

다음은 jeus-web-dd.xml의 보안 설정 예제이다.

웹 모듈 보안 설정 : <jeus-web-dd.xml>

```
<?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>
    <role-permission> ①
      <principal>user1</principal>
      <principal>user2</principal>
      <principal>customerGroup</principal>
      <role>R1</role>
    </role-permission>
  </role-mapping>
</jeus-web-dd>
```

```

        <role-permission> ❷
            <role>R2</role>
            </excluded>
        </role-permission>
        <role-permission> ❸
            <role>R3</role>
            </unchecked>
        </role-permission>
        <role-permission> ❹
            <principal>tellerGroup</principal>
            <role>R4</role>
            <actions>09:00-17:00</actions>
            <classname>
                jeus.security.resource.TimeConstrainedRolePermission
            </classname>
        </role-permission>
    </role-mapping>
<servlet> ❺
    <servlet-name>HelloWorld</servlet-name>
    <run-as-identity>
        <principal-name>user1</principal-name>
    </run-as-identity>
    . . .
</servlet>
    . . .
</jeus-web-dd>

```

위 예제의 각 부분에 대한 의미는 다음과 같다.

- ❶ "user1", "user2", "customerGroup" Principal은 "R1" Role에 속한다.
- ❷ "R2" Role은 Excluded되어 있다. 즉, 아무도 "R2" Role을 부여받을 수 없다.
- ❸ "R3" Role은 Unchecked되어 있다. 즉, 누구나 "R3" Role을 부여받을 수 있다.
- ❹ "tellerGroup" Principal은 오전 9시부터 오후 5시까지 Role "R4"를 부여받는다.
jeus.security.resource.TimeConstrainedRolePermission 클래스의 implies() 메소드에서 구현된다.
- ❺ run-as-principal로 "user1"이 사용되었다. 즉, 서블릿에서 EJB를 호출할 때 "user1"이 security identity로 넘겨진다. Deployer는 "user1"이 web.xml에 정의된 "R1" Role에 속하는지 반드시 확인해야 한다.

3.4. Jakarta EE 애플리케이션 보안 설정

본 절에서는 Jakarta EE 애플리케이션(EAR 파일)의 보안을 설정하는 방법을 설명한다.

기본적인 설정 방법은 다음과 같다.

- [application.xml 설정](#) : 논리적 Role을 선언
- [jeus-application-dd.xml 설정](#) : deploy할 보안 도메인과 Principal-to-Role 매핑을 설정

3.4.1. application.xml 설정

Jakarta EE 애플리케이션은 .ear 확장자를 가진 Archive 파일이다. EAR Archive는 EJB 모듈, 웹 모듈, Connector 모듈 그리고 모듈에서 참조하고 있는 클래스들로 구성되어 있다. META-INF 디렉터리는 application.xml 설정 파일을 포함하고 있는데, 전체 애플리케이션에 대한 여러 가지 정보를 제공한다.

본 절에서는 application.xml에서 보안과 관련된 부분을 설명한다.

기본적으로 application.xml에는 보안과 관련된 설정은 논리적 Role을 설정하는 단 한 가지 밖에 없다. Role 선언은 EAR 파일 내에 있는 모든 EJB 모듈과 웹 모듈에 공통으로 적용되며 애플리케이션 범위(scope)를 가진다.

Jakarta EE 애플리케이션 보안 설정 : <application.xml>

```
<?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>
  <security-role> ①
    <role-name>Administrator</role-name>
  </security-role>
  <security-role> ②
    <role-name>Customer</role-name>
  </security-role>
</application>
```

위의 예제에서는 2가지 Role이 선언되었다.

- ① "Administrator"
- ② "Customer"

두 가지 Role은 Role-to-Resource 매핑을 정의하기 위해 EJB 모듈과 웹 모듈의 DD 파일에서 사용된다. EJB 모듈과 웹 모듈 보안에 대해서는 [EJB 모듈 보안 설정](#)과 [웹 모듈 보안 설정](#)에서 이미 설명되었다.

애플리케이션을 JEUS 서버에 deploy하기 전에 논리적 Role이 실제 Principal에 매핑되었는지 확인해야 한다. 이 매핑은 각각의 경우에 따라 아래 파일에 매핑된다.

- 애플리케이션의 경우 : jeus-application-dd.xml
- EJB 모듈의 경우 : jeus-ejb-dd.xml (Principal-to-Role 설정은 [jeus-ejb-dd.xml 설정](#)을 참고)
- 웹 모듈의 경우 : jeus-web-dd.xml (Principal-to-Role 설정은 [jeus-web-dd.xml 설정](#)을 참고)

애플리케이션 전체에서 Role과 Principal-to-Role 매핑을 공유하기 때문에 Jakarta EE 애플리케이션에 포함되어 있는 모든 모듈을 잘 알고 있어야 한다. 가령, application.xml에 "Administrator"라는 Role을 설정해 놓고, jeus-application-dd.xml에서 "user2"라는 Principal을 할당했다고 가정한다. 이 Role 매핑은 애플리케이션에 포함된

모든 EJB 모듈과 웹 모듈에서 적용된다.

비슷하게 Role과 Principal-to-Role 매핑을 특정 EJB의 jeus-ejb-dd.xml에만 정의했다고 해도 설정된 Role과 Principal-to-Role 매핑은 .ear에 포함된 모든 모듈에서 여전히 공유될 수 있다.



Jakarta EE 애플리케이션에서 모든 Principal-to-Role 매핑은 그것이 jeus-application-dd.xml에 설정되어 있든 특정 jeus-ejb-dd.xml이나 jeus-web-dd.xml에 설정되어 있든 상관없이 애플리케이션 범위(scope)를 가진다는 점을 주의한다.

3.4.2. jeus-application-dd.xml 설정

애플리케이션 레벨에서 Principal-to-Role 매핑은 **jeus-application-dd.xml**에 설정한다.

Principal-to-Role 매핑을 정의하기 위해 jeus-application-dd.xml의 <application> 태그 아래에 다음 태그들을 추가한다.

- <role-permission> (선택)

<role-permission> 태그는 Principal-to-Role 매핑을 정의하며, 선택적으로 Excluded, Unchecked Role을 설정할 수 있다. 이 태그는 jeus-web-dd.xml, jeus-ejb-dd.xml, policies.xml과 완전히 동일하게 동작한다.

다음의 하위 태그를 가진다.

태그	설명
<principal> (0개 이상)	Role에 매핑되는 Principal 이름이다.
<role> (1개, 필수)	Role 이름을 나타낸다. 이는 Role Permission을 구현하는 Java 클래스 생성자에 첫 번째 파라미터로 넘겨진다. Role 이름은 application.xml에 선언된 논리적 Role 이름과 일치해야 한다.
<actions> (선택)	Role Permission에 추가적인 정보를 제공한다. 설정되어 있다면 Role Permission을 구현하는 Java 클래스 생성자에 두 번째 파라미터로 넘겨진다.
<classname> (선택)	Role Permission을 구현하고 있는 Java 클래스명이다. 이 클래스는 java.security.Permission 클래스의 서브 클래스로 최소한 하나의 파라미터(String rolename)를 넘겨받는 public 생성자가 있어야 한다. 생략되어 있다면 기본값으로 jeus.security.resource.RolePermission이 사용된다.
<excluded> (선택)	empty 태그로 설정되어 있으면 Role은 Excluded로 취급된다. 즉, 아무도 해당 Role을 부여받을 수 없다. <principal>이나 <unchecked> 태그보다 우선순위가 높다.
<unchecked> (선택)	empty 태그로 설정되어 있으면, Role이 Unchecked로 취급된다. 즉, 누구나 해당 Role을 부여받을 수 있다. <principal> 태그보다 우선순위가 높다.

<application> 태그 아래에 해당 애플리케이션이 deploy되는 보안 도메인을 설정해 놓아야 한다. 기본적으로 'SYSTEM_DOMAIN'이 사용되지만, 특정 애플리케이션이 특별한 보안 서비스를 필요로 한다면 해당 보안 서비스를 포함하고 있는 새로운 도메인을 생성해서 deploy해야 한다.

jeusadmin의 **deploy** 명령어에서 보안 도메인 이름을 설정할 수 있다.

Jakarta EE 애플리케이션 보안 설정 : <jeus-application-dd.xml>

```
<application>
  . . .
  <role-permission> ①
    <principal>user2</principal>
    <role>Administrator</role>
  </role-permission>
  <role-permission> ②
    <role>Customer</role>
    </unchecked>
  </role-permission>
  . . .
</application>
```

위의 예제에서, Jakarta EE 애플리케이션 보안을 다음과 같이 설정했다.

- ① "user2"라는 Principal은 "Administrator" Role에 매핑되어 있으며, 디폴트 RolePermission을 사용한다.
- ② 모든 Principal에 "Customer" Role을 부여한다(Role이 unchecked되었기 때문이다).



1. 보안 도메인을 설정하는 방법은 [보안 도메인 설정](#)를 참고한다.
2. Custom Permission을 개발하고 설정하는 방법은 [보안 시스템 정책 설정](#)을 참고한다.
3. JEUS에 Jakarta EE 애플리케이션을 deploy하는 방법은 "JEUS Server 안내서"를 참고한다.

3.5. 예제

본 절에서는 위에서 설명한 설정을 기반으로 작성한 예제에 대해 간략하게 설명한다.

다음 화면은 최초로 접속할 때의 로그인 화면이다. 해당 URL에 checked permission이 설정된 경우에만 보이며, Unchecked permission일 때는 이 화면이 생략되고 바로 페이지가 표시된다. Excluded permission일 때는 Authorization이 실패하고 에러 화면이 나타난다.

[home](#)

Login

Welcome! This page gives information of the security configuration for applications (EJBs and Servlets). Please login to access the main page.

Username:

Password:

Login

로그인 화면

인증에 실패하였거나 권한이 없는 계정으로 로그인했을 경우 에러 페이지가 나타난다. 권한이 있는 계정으로 로그인에 성공하면 다음과 같은 화면이 나타난다.

[home](#)

JEUS Security Sample

The following samples are offered to show how the security configuration works. Each sample's target resource is a Web URL or an EJB method.

Servlet Sample

servlet-name	authorization	servlet-class
unchecked	unchecked	webpages.JoinUs
checked	checked	webpages.OnlyOne
excluded	excluded	webpages.WithYou

EJB Sample

method-name	authorization	ejb-name
message	unchecked	ejbwork.PublicCall
message	checked	ejbwork.Admin
excludedMSG	excluded	ejbwork.Secret
callsecret	applied run-as	ejbwork.PublicCall

메인 화면

이 예제는 메인 페이지에서 3개의 서블릿으로 링크가 걸려 있으며, 3개의 EJB 메소드를 호출하게 된다. 로그인한 계정과 설정 파일에 따라 접근할 수 있는지 여부에는 차이가 있으므로 여러 가지 설정을 하고 여러 계정으로 로그인해 보면서 직접 확인해본다.

4. 보안 시스템 API 프로그래밍

본 장에서는 보안 시스템 API 프로그래밍에 대해 설명한다.

4.1. 개요

사용자 애플리케이션에 자신만의 특별한 보안 기능을 추가하기 위해 보안 시스템 API를 사용하여 프로그래밍할 수 있다. 이러한 예로는 애플리케이션을 통해 등록한 사용자를 자동으로 JEUS 보안 시스템에 등록하는 `RegistrationServlet`("auto-registration"이라고 한다)이 있다.

애플리케이션 프로그래머는 보안 서비스를 개발하기 전에 표준 Jakarta EE 보안 모델과 JEUS의 보안 서비스들이 원하는 보안 기능을 제공하는지 먼저 확인한다. 보안 API를 사용하여 프로그램을 개발하게 되면 Jakarta EE 서버 간의 호환성이 떨어진다. 일반적으로 Jakarta EE 서버 간의 호환성을 유지하기 위해 표준 Jakarta EE 보안 인터페이스만 사용하기를 권장한다.

4.2. Java SE Permission 설정

악의적인 사용자 코드(서블릿, EJB)로부터 JEUS 시스템을 보호하기 위해 보안 API를 사용할 수 있다.

사용자 코드에 보안 API를 사용하는 경우는 Java SE `SecurityManager`를 사용하거나 소스 코드(서블릿, EJB)가 `SecurityCommonService.loginDefault(Subject)`를 사용하여 성공적으로 로그인한 경우이다. 이때 Subject는 Target 보안 도메인의 `accounts.xml`에 미리 설정되어 있는 사용자에 대한 Subject로 `policies.xml`에 설정된 필요한 JEUS Permission을 가지고 있는 경우이다.

각 파일에 대한 설정 방법은 다음을 참고한다.

- Java SE `SecurityManager`와 Java SE Policy 파일 : [Java SE SecurityManager 설정](#)
- `accounts.xml` : [보안 시스템 사용자 정보 설정](#)
- JEUS 보안 시스템에 `policies.xml` : [보안 시스템 정책 설정과 참고 자료](#)

4.3. 기본 API

애플리케이션 프로그래밍 레벨에서 보안 시스템과 연동할 때 **`jeus.security.base`** 패키지의 클래스는 중요한 역할을 한다.

클래스	설명
<code>jeus.security.base.Subject</code>	사용자를 나타낸다. Subject는 단 하나의 메인 Principal을 가지고 있으며, 메인 Principal이 Subject의 ID(username)로 취급된다. <code>jeus.security.base.Subject</code> 클래스에 여러 개의 String 속성값이 전달되기도 한다.

클래스	설명
jeus.security.base.CredentialFactory	Subject 클래스의 멤버 변수이며, Subject에 대한 실제 Credential을 생성하는 데 사용된다. 예를 들어 PasswordFactory 클래스는 패스워드 Credential 인스턴스를 생성하고, JKSConnectionFactory 클래스는 JKS Keystore로부터 인증서를 얻어와서 인증서 Credential 인스턴스를 생성한다.
jeus.security.base.Policy	하나의 Principal-to-Role 매핑과 여러 개의 Role-to-Resource 매핑을 나타낸다. PermissionMaps을 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 Role을 나타내는 인터페이스이다. Role-to-Resource PermissionMap에서는 Role 인스턴스가 Resource Permission에 매핑된다. 마찬가지로 Principal-to-Role PermissionMap에서는 Principal이 Role Permission에 매핑된다.
jeus.security.base.SecurityCommonService	해당 Subject를 인증하고, Subject에 대한 Permission을 체크한다.
jeus.security.base.SecurityException	로그인 실패, 인증 실패, 권한 체크 실패 등의 보안을 위반했을 때 발생하는 예외이다.
jeus.security.base.ServiceException	보안 시스템에서 심각한 런타임 에러가 발생할 때 발생하는 예외이다.

4.4. 리소스 API

리소스와 관련해서 jeus.security.base 패키지의 몇 가지 기본 클래스뿐만 아니라 **jeus.security.resource** 패키지의 클래스도 중요한 역할을 한다.

클래스	설명
jeus.security.resource.PrincipalImpl	java.security.Principal 인터페이스의 구현 클래스이다.
jeus.security.resource.GroupPrincipalImpl	그룹 Principal을 나타내는 PrincipalImpl의 서브 클래스로 Nested한 그룹의 멤버를 관리하는 java.security.acl.Group 인터페이스의 구현 클래스이다.
jeus.security.resource.Password	단순한 Password Credential 인스턴스로 PasswordFactory에 의해 생성된다.
jeus.security.resource.PasswordFactory	Password Credential을 생성하는 CredentialFactory이다.
jeus.security.resource.Lock	일종의 Credential로 해당 Subject에 Lock을 건다. Lock이 걸린 Subject로 로그인하게 되면 항상 실패하게 된다.
jeus.security.resource.LockFactory	Lock Credential을 생성하는 CredentialFactory이다.

클래스	설명
jeus.security.resource.ExpiryTime	일종의 Credential로 해당 Subject의 만료 기간을 설정한다. Subject가 만료된 후 해당 Subject를 사용하여 로그인을 시도하면 모두 실패한다.
jeus.security.resource.ExpiryTimeFactory	ExpiryTime을 생성하는 CredentialFactory이다.
jeus.security.resource.RoleImpl	Role 인터페이스를 구현한 클래스이다.
jeus.security.resource.RolePermission	특정 Principal이 특정 Role에 속한다는 것을 나타내는 java.security.Permission의 서브 클래스이다. RolePermission은 Principal-to-Role 매핑을 나타낼 때 사용된다.
jeus.security.resource.TimeConstrainedRolePermission	RolePermission의 서브 클래스로 현재 시간이 설정된 시간 범위 내에 있을 때만 Principal이 이 클래스의 슈퍼 클래스가 나타내는 Role에 속한다는 것을 나타낸다.
jeus.security.resource.ResourcePermission	java.security.Permission의 서브 클래스로 Role이 리소스에 접근해서 특정 액션을 실행할 수 있다는 개념을 나타낸다. 따라서 ResourcePermission은 Role-to-Resource 매핑을 나타낼 때 사용된다.



해당 클래스에 대한 자세한 정보는 Javadoc과 [참고 자료](#)를 확인한다.

4.5. SPI 클래스

보안 시스템의 근간을 이루는 서비스와 작업하려면, **jeus.security.spi** 패키지의 SPI 클래스를 사용해야 한다.

클래스	설명
jeus.security.spi.AuthenticationRepositoryService	Subject 저장소로부터 Subject를 추가, 삭제, 조회하는 데 사용된다. 이 API를 이용하면 Subject(user)를 프로그램 내에서 추가할 수 있다.
jeus.security.spi.AuthorizationRepositoryService	Policy 저장소로부터 Policy 데이터를 추가, 삭제, 조회하는 데 사용된다. 이 API를 이용하면 프로그램 내에서 Permission을 추가할 수 있다.



해당 클래스에 대한 자세한 정보는 Javadoc과 [참고 자료](#)를 확인한다. SPI 클래스에 대한 더욱 상세한 정보는 [Custom 보안 서비스 개발](#)을 참고한다.

4.6. 예제

다음 예제는 보안 API를 사용한 프로그램의 일부이다.

```
// Login the CodeSubject so that security checks are
```

```

// disabled (so that we can modify the Subject and Policy
// stores)
SecurityCommonService.loginCodeSubject();

// Make Subject with Principal "pete"
Principal petePrincipal = 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
SecurityCommonService.logout();

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

// Login Subject "pete" (should succeed since we added
// "pete" earlier)
SecurityCommonService.loginDefault(pete2);

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

// Print the name of the current Subject ("pete")
System.out.println(

```

```
SecurityCommonService.getCurrentSubject().getPrincipal().getName());
```

```
// Logout "pete"
```

```
SecurityCommonService.logout();
```

5. Custom 보안 서비스 개발

본 장에서는 JEUS 보안 시스템의 주요 특징인 Custom 보안 서비스를 개발하는 방법에 대해 설명한다.

5.1. 개요

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

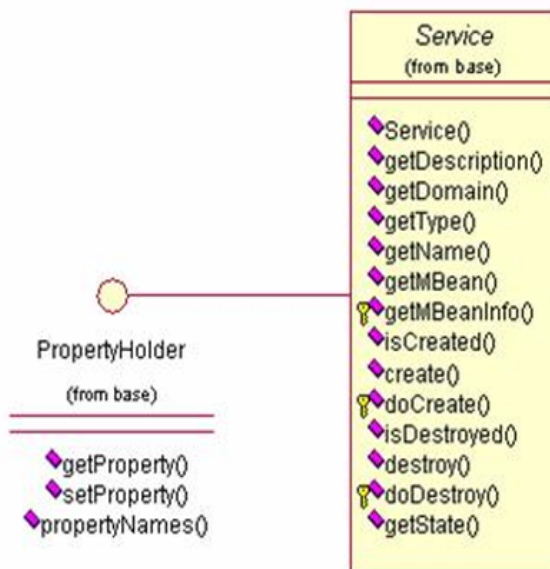
다음은 Custom 보안 서비스를 개발과 관련된 몇 가지 중요한 개념으로 각 절에서 상세히 설명한다.

- 서비스 클래스
- 구현 기본 패턴
- 11개의 SPI 클래스
- Custom 보안 서비스 설정 방법

5.2. 서비스 클래스

pluggable 보안 아키텍처에서 가장 기본이 되는 클래스는 `jeus.security.base.Service` 클래스이다. 서비스 클래스는 보안 서비스를 구현하려는 클래스들이 반드시 확장해야 하는 추상 클래스로 현재 `jeus.security.spi` 패키지에 있는 모든 SPI 클래스도 이 클래스를 확장한 것이다.

다음은 서비스 클래스의 클래스 다이어그램이다.



서비스 클래스 다이어그램

서비스 클래스는 모든 보안 서비스에 공통적으로 적용되는 다음의 항목들을 포함하고 있다.

- Service Description

서비스 클래스는 제공하려는 서비스가 무엇인지에 대해 간단하게 설명하는 String 타입의 description을 제공한다. 서비스에 대한 설명은 **getDescription()** 메소드로 가져올 수 있다.

- **Service Domain**

런타임에 생성된 인스턴스는 특정 도메인에 할당된다. 따라서 도메인은 본질적으로 서비스 인스턴스의 집합이라 할 수 있다. 서비스의 도메인은 **getDomain()** 메소드로 가져올 수 있다.

- **Service Type**

서비스 구현 클래스는 단 하나의 타입을 가진다. 타입이란 Service set으로부터 특정 서비스 인스턴스를 가져오기 위해 사용하는 일종의 마크이다.

예를 들어 특정 서비스 구현 클래스가 인증 기능을 제공한다면 다른 보안 서비스에서 이 인증 서비스 클래스를 사용하기 위해 도메인에 해당 클래스를 요청해야 한다. 이때 서비스 타입을 전달해서 도메인이 정확한 서비스 인스턴스를 찾아서 리턴하도록 만들 수 있다.

서비스 타입은 **getType()** 메소드로 가져올 수 있다. 그러나 **getType()** 메소드는 추상 메소드이므로 반드시 서브 클래스에서 구현해야 한다. 서비스 타입은 실제로는 java.lang.Class 타입으로 jeus.security.spi 패키지에 포함되어 있는 SPI 클래스의 클래스 인스턴스이다. 각 SPI 클래스에서 **getType()** 메소드를 final로 구현하고 있다. 즉, SPI의 서브 클래스에서는 **getType()** 메소드를 재정의할 수 없다.

- **Service**

서비스의 명칭은 **getName()** 메소드로 가져올 수 있다.

- **Service MBean**

서비스 클래스는 특정 서비스 인스턴스를 관리하는 데 사용되는 JMX MBean과 결합되어 있다.

MBean은 **getMBean()** 메소드로 가져올 수 있으며, 서브 클래스에서 디폴트 MBean이 아닌 다른 MBean을 리턴하려면 이 메소드를 재정의해야 한다.

기본적으로 MBean은 protected 메소드로 선언된 **getMBeanInfo()**로부터 리턴된 MBeanInfo를 바탕으로 생성된다. 서브 클래스는 이 메소드를 재정의해서 디폴트 이외의 다른 MBeanInfo 인스턴스를 리턴하도록 할 수 있다. 메소드를 재정의할 때 슈퍼 클래스의 MBeanInfo가 누락되지 않도록 주의한다.

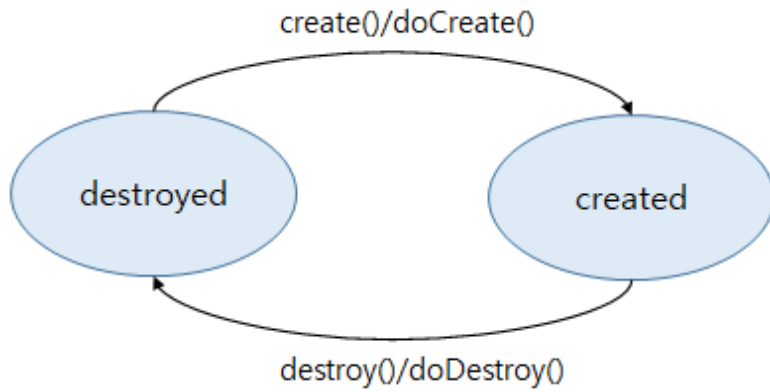
- **Service State**

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

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

서비스 클래스의 현재 상태는 **isCreate()**와 **isDestroyed()** 메소드를 호출하여 알 수 있다. 또한 현재 상태를 String으로 리턴하는 **getState()** 메소드를 가지고 있다.

다음은 서비스 클래스의 상태 차트에 대한 그림이다.



서비스 클래스의 상태 차트

• Service Properties

서비스 클래스는 name-value 쌍의 속성을 가지고 있다. 이러한 속성을 설정하고 가져오는 작업은 `jeus.security.base.PropertyHolder` 인터페이스를 통해 이루어진다. 서비스 구현 클래스는 이 인터페이스를 구현해야 한다.

속성들은 서비스 인스턴스를 초기화하는 데 사용된다. 서비스 인스턴스가 생성되고 `create()` 메소드가 불려지기 전에 속성들이 설정된다. 이후 `doCreate()` 메소드가 불려지면 설정된 속성값을 Query해 와서 Service 인스턴스를 적절하게 초기화한다.

5.3. Custom 보안 서비스 구현 패턴

본 절에서는 Custom 보안 서비스의 구현 방법에 대해서 간략하게 설명한다.

일반적인 Custom 보안 서비스를 구현하는 과정은 다음과 같다.

1. 사전에 보안 시스템과 보안 시스템 아키텍처를 충분히 이해하고 있어야 한다.
2. Custom 보안 서비스가 어떤 보안 기능을 제공해야 하는지 파악한다.
3. 해당 특성을 가지고 있는 SPI 클래스를 선택한다. 주요 SPI 클래스에 대한 자세한 내용은 [SPI 클래스](#)를 참고한다.
4. 해당 SPI 클래스에 대한 문서를 주의 깊게 살펴본다(Javadoc, [참고 자료](#), [SPI 클래스](#)에 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 서비스 초기화에 사용되는 속성들을 정의한다. 각 속성은 `public static final String` 타입으로 각각이 무엇을 나타내는지는 문서화되어야 한다.
 - b. `doCreate()` 메소드를 구현한다. 이 메소드는 보안 서비스가 시작할 때 단 한 번 불리며, 자원 할당과 같은 일반적인 초기화 작업을 수행한다. 이 메소드는 `getProperty()` 메소드를 통해 설정값을 읽어 들인다. `getProperty()` 메소드의 파라미터는 이전 단계에서 설정된 속성명이다.
 - c. `doDestroy()` 메소드를 구현한다. 이 메소드는 서비스가 종료하기 전에 단 한 번 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 Javadoc에서 설명하고 있는 방식대로 구현한다.
 - e. 선택적으로 JMX를 통해 서비스를 관리하는 데 사용되는 메소드들을 구현할 수 있다. `getMBean()` 또는

getMBeanInfo() 메소드를 구현한다.

6. SPI를 구현한 클래스를 컴파일한다.

7. [보안 시스템 설정](#)에서 설명한 대로 새로운 보안 서비스를 JEUS에 등록한다.

5.4. SPI 클래스

본 절에서는 jeus.security.spi 패키지에 포함되어 있는 다양한 SPI 클래스에 대해 설명한다.

다음은 jeus.security.spi 패키지 내에 정의된 SPI 클래스 목록이다. Cardinality는 각 도메인에 해당 SPI 클래스가 몇 개나 있을 수 있는지에 대한 설명이다. SPI 클래스에 대한 상세한 설명은 [참고 자료](#)를 참고한다.

class name	Purpose	Cardinality
SecurityInstaller	보안 시스템을 설치하고 제거한다. 전체 JVM에 대해 오직 하나만 존재해야 한다.	1
SubjectValidationService	로그인 전에 해당 Subject의 Credential이 유효한지 아닌지 체크한다.	0개 이상
SubjectFactoryService	로그인 전에 Custom Subject를 생성한다.	1
AuthenticationService	로그인 전에 Subject를 인증한다.	1
AuthenticationRepositoryService	Subject를 Subject 저장소로부터 추가, 삭제, 조회한다.	1
IdentityAssertionService	cert-user-map.xml 파일 정보를 이용하여 Credential을 Subject에 매핑한다.	0개 이상
CredentialMappingService	Truststore 정보를 이용하여 Credential을 Subject에 매핑한다.	0개 이상
CredentialVerificationService	Subject가 가지고 있는 Credential 중 최소한 하나라도 유효한지 검증한다.	0개 이상
AuthorizationService	Subject가 특정 Role 또는 리소스에 접근할 권한이 있는지 체크한다.	1
AuthorizationRepositoryService	Policy를 저장소로부터 추가, 삭제, 조회한다.	1
EventHandlingService	보안 이벤트에 대한 Custom 이벤트 핸들러, 다양한 보안 감사를 구현한다.	0개 이상

5.4.1. SubjectValidationService SPI

jeus.security.spi.SubjectValidationService SPI는 Subject가 가지고 있는 Credential이 유효한지 아닌지를 체크하는 데 사용된다. 이것은 때때로 Credential이 유효하지 않는 경우도 있음을 나타낸다. 유효하지 않는 Credential은 곧 Subject가 유효하지 않다는 의미이고, 결과적으로 해당 Subject로는 로그인할 수 없다는 것을 나타낸다.

SubjectValidationService의 전형적인 예는 Subject가 "lock" Credential을 가지고 있는지 여부를 체크하는

것이다. 만약 "lock" Credential을 가지고 있다면, Subject는 Lock에 걸린 것으로 취급되어 더 이상의 로그인 프로세스가 진행되지 않는다.

SubjectValidationService.checkValidity(Subject) 메소드는 보통 로그인 과정에서 호출된다. 만약 이 메소드에서 SecurityException이 발생되면, 모든 로그인 과정은 실패로 돌아간다. Subject에 "lock" Credential을 자동으로 설정하기 위해서 EventHandlerService가 사용된다. 이에 관한 자세한 설명은 [EventHandlerService SPI](#)를 참고한다.

인증(파라미터로 넘어온 Subject가 실제 해당 Subject인지 검증하는 것)과 Subject 유효성 검사는 서로 별개의 것이나 로그인 과정에는 2가지 서비스가 모두 사용된다.

도메인당 0개 이상의 SubjectValidationService 인스턴스가 설정될 수 있다. 전체 SubjectValidationService에서 SecurityException이 하나도 발생하지 않으면, Subject는 유효하다고 판단되고, 로그인 과정이 계속 진행된다. 그러나 만약 SecurityException이 한 곳이라도 발생하면 전체 유효성 검사는 실패로 돌아가고 로그인 과정은 더 이상 진행되지 않는다.



SubjectValidationService SPI는 CredentialVerificationService와 많은 점에서 비슷해 보이지만 기능명에서 다르다.

SubjectValidationService는 Credential을 체크해서 Subject가 유효한지 아닌지를 판단하고, CredentialVerificationService는 해당 Subject가 실제 Subject인지 증명하는 Credential을 최소한 하나라도 가지고 있는지를 체크한다. 따라서, SubjectValidationService는 Subject가 성공적으로 인증된 후 사용되고, 반면에 CredentialVerificationService는 AuthenticationService에서 인증 과정 중에 사용된다.

5.4.2. SubjectFactoryService SPI

jeus.security.spi.SubjectFactoryService SPI 클래스는 외부에서 제공하는 정보없이 Subject를 생성하는 특별한 SPI이다.

SubjectFactoryService SPI는 보통 어떤 Subject나 Credential도 파라미터로 넘겨받지 않고, Subject를 SubjectFactoryService를 통하여 생성할 때 사용된다. 예를 들어 전형적인 구현 클래스는 프롬프트로부터 사용자명과 패스워드를 입력받아 이 정보를 기초로 Subject를 생성해서 리턴한다.

SubjectFactoryService는 로그인 메커니즘에서 패스워드 이외의 Credential 타입도 지원하기 위해 만들어졌다.

5.4.3. AuthenticationService SPI

jeus.security.spi.AuthenticationService는 Subject를 인증하기 위한 클래스이다. 즉, 파라미터로 넘겨받은 Subject가 Subject 저장소에 등록되어 있는 실제 Subject와 일치하는지 검증하는 것이다.

인증 과정은 다음과 같다.

1. jeus.security.spi.CredentialMappingService.getSubjectName(Object) 메소드를 호출하여 Subject의 Credential이 어떤 사용자에게 매핑되는지 확인한다.
2. jeus.security.spi.AuthenticationRepository.getSubject(String username) 메소드를 호출하여 Subject

저장소로부터 실제 등록되어 있는 Subject를 로컬로 복사해온다. 이를 로컬 Subject라고 한다.

3. 만약 로컬 Subject가 null이 아니라면 로컬 Subject의 Credential과 인증할 Subject의 Credential이 일치하는지 비교한다. 이 비교는 `jeus.security.spi.CredentialVerificationService.verifyCredentials(Subject, Subject)` 메소드를 호출하여 수행된다.

몇몇 경우에는 `equals(Object)`를 사용하여 Credential을 비교하기도 하는데, 이는 유연한 방법이 아니다.

4. 위의 과정이 모두 성공적으로 완료되면 로컬 Subject가 `authenticate(Subject)` 메소드로부터 리턴되고, 결국 이 Subject를 사용하여 `SecurityCommonService`로 로그인한다.

도메인당 1개 이상의 `AuthenticationService`를 설정할 수 있다. 만약 설정된 `AuthenticationService` 중 최소한 하나라도 Subject를 성공적으로 인증했다면 모든 `AuthenticationService`가 인증한 것으로 간주한다. 즉, 단 하나의 `AuthenticationService`라도 성공적으로 Subject를 인증했다면, 추가적으로 다른 `AuthenticationService`에 Query를 던지지 않는다.

5.4.4. AuthenticationRepositoryService SPI

`jeus.security.spi.AuthenticationRepositoryService` SPI는 Subject 저장소로부터 Subject를 추가, 삭제, 조회하는 메소드를 가지고 있다. 이 SPI는 `AuthenticationService`를 구현하는 클래스에서 사용되거나, Jakarta EE 애플리케이션 코드 내에서 직접 사용될 수 있다.

예를 들어 웹 사이트를 통해 새로운 사용자가 등록할 때 마다, 자동으로 Subject를 Subject 저장소에 추가하는 코드를 작성하려면, 이 SPI를 사용해야 한다.

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

일반적으로 `AuthenticationRepositoryService` SPI를 별도로 구현할 필요는 없다. 그러나 디폴트 `AuthenticationService`를 사용하거나(사용자 인증 과정에서 `AuthenticationRepositoryService.getSubject(String)` 메소드를 호출하기 때문이다), Jakarta EE 애플리케이션 코드에서 직접 사용할 때는 반드시 구현해야 한다.

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

5.4.5. IdentityAssertionService SPI

`jeus.security.spi.IdentityAssertionService` SPI는 Subject의 사용자명(username)이 제공되지 않는 경우에 필요하다. 즉, 메인 Principal이 null일 때이다. 이 경우에는 Subject로부터 Credential을 가져와서 `cert-user-map.xml`의 정보를 기반으로 Credential이 어느 사용자명에 매칭되는지 알아보게 된다. 매칭된 사용자명은 이후 인증과정에 사용된다.

전형적인 예로 `java.security.Certificate` 인스턴스를 들 수 있다. 이 인스턴스는 `Certificate Credential`을 사용하므로 메인 Principal을 가지고 있지 않다. 이 경우 인증과정 동안 Subject로부터 `Certificate Credential`을 얻어와서 `IdentityAssertionService.getIdentity(Object)` 메소드에 파라미터로 넘긴다.

해당 메소드는 다시 IdentityAssertionService.doIdentity(Object)를 호출한다. 이 메소드 내부에서 cert-user-map.xml에 정의된 Certificate에 Attribute Key값에 대응하는 사용자명을 역추적하고 결국 매칭되는 사용자명을 리턴한다.

5.4.6. CredentialMappingService SPI

jeus.security.spi.CredentialMappingService는 Subject의 사용자명(username)이 제공되지 않는 경우에 필요하다. 즉, 메인 Principal이 null일 때이다. 이 경우에는 Subject로부터 Credential을 가져와서 Credential이 어느 사용자명에 매칭되는지 알아보게 된다. 매칭된 사용자명은 이후 인증 과정에 사용된다.

전형적인 예로 java.security.Certificate 인스턴스를 들 수 있다. 이 인스턴스는 Certificate Credential을 사용하므로 메인 Principal을 가지고 있지 않다. 이 경우 인증과정 동안 Subject로부터 Certificate Credential을 얻어 와서, CredentialMappingService.getSubjectName(Object) 메소드에 파라미터로 넘긴다.

해당 메소드는 다시 CredentialMappingService.doGetSubjectName(Object)를 호출한다. 이 메소드 내부에서 Certificate 저장소로부터 Certificate를 통해 사용자명을 역추적하고, 결국 매칭되는 사용자명을 리턴한다.

5.4.7. CredentialVerificationService SPI

jeus.security.spi.CredentialVerificationService SPI는 새로운 타입의 Proof Credential을 지원하기 위해 사용된다.

Proof Credential는 파라미터로 넘어온 Subject가 실제 존재하는 Subject인지 검증하기 위해 사용되는 Credential이다. Proof Credential의 전형적인 예가 바로 패스워드로 jeus.security.resource.Password 클래스로 구현된다.

CredentialVerificationService SPI는 서브 클래스가 반드시 구현해야 하는 단 하나의 메소드인 doVerifyCredentials(Subject, Subject)를 선언하고 있다. 이 메소드 Signature에서 첫 번째로 나오는 Subject는 "reference Subject"라고 하고, Subject 저장소에 등록되어 있는 실제 Subject의 Credential들을 가지고 있다. 두 번째로 나오는 Subject는 "proof Subject"라고 하고, "proof Credential"들을 가지고 있다. doVerifyCredentials(Subject, Subject) 메소드는 2개의 Subject의 Credential들을 서로 비교해서 하나라도 일치하는 것이 있는지 확인한다.

Credential들 간의 매칭은 다양한 방법으로 이루어진다(보통 equals(Object) 메소드만으로는 충분하지 않다). 만약 두 Subject 간에 하나라도 일치하는 Credential이 있다면 메소드는 리턴되고, 그렇지 않다면 jeus.security.base.SecurityException이 발생한다.



몇몇 경우에는 Reference Subject에 대한 정보가 필요 없을 때가 있다. 즉, 특정 Proof Credential의 경우 Proof Subject에 대한 정보만 가지고 Credential을 검증할 수 있다. 예를 들어 특정 Certificate Credential의 경우가 그렇다.

jeus.security.resource.Password를 매치하는 CredentialVerificationService는 jeus.security.impl.verification.PasswordVerificationService 클래스이다.

도메인당 0개 이상의 CredentialVerificationService를 설정할 수 있다. 만약 최소한 하나의 CredentialVerificationService에서라도 매칭이 확인되면 전체 CredentialVerificationService가 모두 성공한

것으로 간주된다. 그렇지 않으면 `SecurityException`이 발생하고, 전체 검증은 모두 실패한 것으로 간주되어 결과적으로 인증 과정이 실패하게 된다.

5.4.8. AuthorizationService SPI

`jeus.security.spi.AuthorizationService` SPI는 보안 시스템에서 권한 체크와 관련된 메소드를 정의하고 있다. 이 SPI의 목적은 "Subject S는 A라는 액션을 실행할 권한이 있는가?"라는 질문에 답하는 것이다.

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` SPI를 사용하지 않는 다른 종류의 구현도 얼마든지 가능하다.

도메인당 1개 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 허가되었다면, 추가적으로 다른 `AuthorizationService`에 권한을 확인하는 Query를 던지지 않는다.

5.4.9. AuthorizationRepositoryService SPI

`jeus.security.spi.AuthorizationRepositoryService` SPI는 정책(Policy) 저장소로부터 `jeus.security.base.Policy` 인스턴스를 추가, 제거, 조회하는 메소드를 포함하고 있다. 이 SPI는 `AuthorizationService` 구현 클래스에서 사용되거나 Jakarta EE 애플리케이션 코드 내에서 특정 이벤트가 발생할 때 직접 정책을 저장소에 추가하거나 제거하기 위해 사용한다.

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

`AuthorizationRepositoryService` SPI를 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 1개 이상의 `AuthorizationRepositoryService`를 도메인에 설정하는 것은 불가능하다.

5.4.10. EventHandlerService SPI

`EventHandlerService` SPI는 각종 보안 이벤트를 캡처해서 처리하는 인터페이스이다.

`EventHandlerService` SPI를 사용하면 보안 이벤트가 발생할 때 로그를 남기고 관리자에 자동으로 메일을 보내고, Subject에 Lock을 설정하는 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 `EventHandlerService`들에 통보된다. 그러면 `EventHandlerService`는 이벤트의 내용에 따라 각각 다르게 처리한다.

`EventHandlerService`의 예로 Subject에 Lock을 거는 경우를 생각해 볼 수 있다. `AuthenticationService`(인증 서비스)가 실패할 때마다 "security.authentication.failed" 타입의 이벤트가 발생한다. 그러면 `logout`

EventHandlingService가 이벤트를 캐치하여, handleEvent(Event) 메소드를 실행시킨다. 이 메소드 내부에서는 로그인 횟수를 카운트하고 있다가 특정 값(예를 들면 3회) 이상에 도달하면 해당 Subject에 "lock" Credential을 설정한다.

이후 SubjectValidationService는 해당 Subject에 Lock이 걸렸음을 확인하고, 결과적으로 이 Subject로의 로그인 시도는 모두 실패하게 만든다. SubjectValidationService와 EventHandlingService를 결합하면, 한번에 3회 이상 로그인에 실패한 Subject에 Lock을 걸어 더 이상 시스템에 로그인 시도를 할 수 없게 만드는 기능을 추가할 수 있다.

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

도메인당 0개 이상의 EventHandlingService를 설정할 수 있다.

5.4.11. Dependencies between SPI Implementations

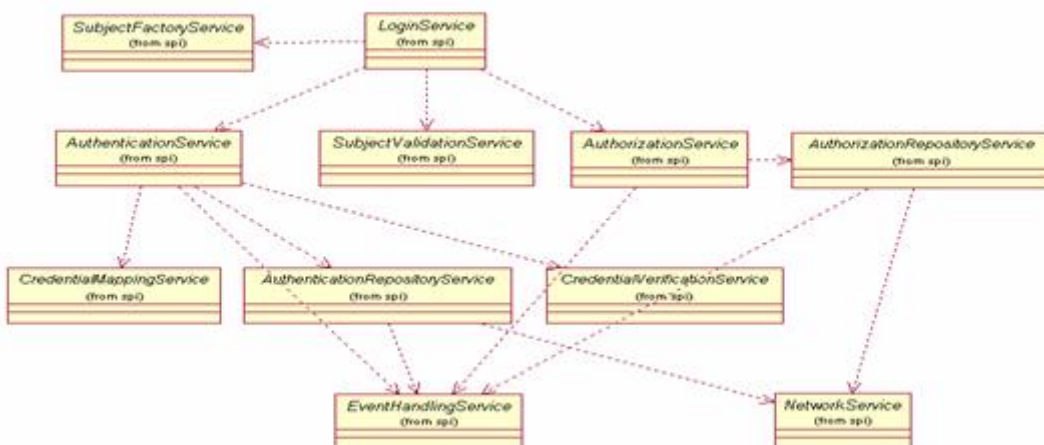
SPI 클래스 간에는 의존이 존재할 수 있다. 예를 들어 AuthenticationService는 AuthenticationRepositoryService에 의존한다.



의존은 하나의 SPI 구현 클래스가 다른 SPI 구현 클래스의 static 메소드를 호출한다는 것을 말한다. AuthenticationService 구현 클래스를 AuthenticationServiceImpl이라 하면 이 클래스는 AuthenticationRepositoryService.getSubject()를 호출하여 Subject에 대한 정보를 얻어온다.

SPI 클래스 간에는 의존이 존재할 수 있는 것은 의존이 존재하지 않을 수도 있다는 것을 의미한다. 이는 SPI 클래스를 어떻게 구현하느냐에 결정된다. 어떤 SPI 클래스도 결코 직접적으로 다른 SPI 클래스의 메소드를 호출하지 않는다 (몇 가지 사소한 예외를 제외함). 일반적으로 SPI 를 구현한 클래스에서 다른 SPI 구현 클래스의 메소드를 호출함으로써 의존이 성립된다.

다음 그림은 디폴트 보안 시스템 구현에서 SPI 구현 클래스 간의 의존에 대한 설명으로 클래스 간의 관계를 단순화시킨 것이다.



기본 보안 시스템 구현에서의 SPI 구현 클래스

5.5. 보안 서비스 설정

새로운 SPI 구현 클래스를 컴파일한 다음, 이 클래스를 JEUS 보안 시스템에 등록해야 한다. 등록하는 방법은 [보안 도메인 구성요소 설정](#)을 참고한다.

6. JACC Provider 사용

본 장에서는 JACC의 목적을 설명하고 Custom JACC Provider를 구현하여 JEUS 보안 시스템에서 사용하는 방법에 대해 간단히 설명한다.

6.1. 개요

Jakarta Authorization는 J2EE 버전 1.4에서 Java Authorization Service Provider Contract for Containers(JACC) 라는 이름으로 처음으로 소개되었고, 현재 버전은 2.0이다.

JACC 배경에는 다음의 2가지 기본 목적이 있다.

- EJB와 서블릿의 권한을 체크하는 경우 표준 SPI를 제공한다.
- 기존 Java SE 보안 모델과 Jakarta EE 보안 모델 간의 조화를 추구한다.

JACC는 EJB와 서블릿의 접근 권한을 정의하고 처리하는 표준적인 방법을 제시하고 있다. 따라서, JACC Provider를 작성하면 JACC 호환 Jakarta EE 서버에서 사용할 수 있다.

6.2. JACC 규약

전체 JACC 스펙은 다음과 같이 3개의 세부 규약으로 구성되어 있다.

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



JACC 스펙에 대한 자세한 정보는 [Jakarta EE Authorization 2.0 스펙 문서](#)를 참고한다.

6.2.1. Provider 설정 규약

JACC Provider 설정 규약은 애플리케이션 서버의 런타임 환경에 JACC Provider를 통합하는 방법을 기술하고 있다. 즉, 이 스펙은 JACC Provider를 Jakarta EE 서버에 등록하는 방법을 정의하고 있다. 일반적으로 이 과정은 매우 단순하다.

1. jakarta.security.jacc.policy.provider 시스템 속성에 Custom java.security.Policy 클래스 이름을 설정한다. JACC Provider는 Custom java.security.Policy를 포함하고 있다.
2. Jakarta EE 서버는 이 클래스명을 읽어서 인스턴스를 만든 다음 java.security.Policy 타입으로 캐스팅한다.
3. 위의 과정이 성공적으로 완료되었다면, java.security.Policy.setPolicy() 메소드를 호출하여 디폴트 Java SE Policy 클래스를 새로 생성한 JACC Policy 클래스로 교체한다.
4. 모든 권한 체크는 새로 등록된 JACC Policy를 통해서 이루어지고 java.security.Policy.getPolicy() 메소드를 통해 현재 JACC Policy를 얻어올 수 있다.



JACC Provider 설정 규약에 대한 자세한 설명은 JACC 스펙을 참고한다.

6.2.2. Policy 설정 규약

Policy 설정 규약은 Jakarta EE DD 파일(ejb-jar.xml, web.xml)에 설정된 보안 제약(security constraints)을 jakarta.security.jacc 패키지에 정의된 java.security.Permission set으로 매핑하는 방법과 이러한 Permission을 JACC Provider(Custom java.security.Policy)에 추가하는 방법을 기술하고 있다.

이는 결국 EJB와 서블릿에서 JACC Provider가 권한 부여와 관련된 결정을 내릴 수 있게 만든다.



Policy 설정 규약은 Principal-to-Role 매핑에 대한 언급이 없다. 오로지 Jakarta EE DD 파일에 근거하여 Role-to-Resource 매핑을 어떻게 할지만 정의하고 있다. 따라서 Principal-to-Role 매핑은 JACC Provider를 공급하는 업체에 따라 각각 다른 방법으로 정의되어 있다.

Policy 설정 과정을 간단하게 설명하면 다음과 같이 구성되어 있다.

1. 시스템 속성(-D 속성으로 설정) jakarta.security.jacc.PolicyConfigurationFactory.provider에 Custom jakarta.security.jacc.PolicyConfigurationFactory 클래스명을 설정한다.

JACC Provider는 Custom jakarta.security.jacc.PolicyConfigurationFactory를 포함하고 있다.

2. Jakarta EE 서버는 이 속성값을 읽어서 해당 클래스의 인스턴스(PCF)를 생성한다.
3. 서블릿과 EJB 모듈을 deploy한다.
4. Deployment 코드는 web.xml과 ejb-jar.xml DD 파일을 읽어들이 설정된 security constraints 항목을 JACC Permission 인스턴스들로 전환한다. 이 Permission 클래스는 jakarta.security.jacc 패키지에 포함되어 있다. 이때 각 Permission 인스턴스는 서블릿과 EJB에 설정되어 있는 Role-to-Resource 매핑을 나타낸다.
5. Deployment 코드는 PCF.getPolicyConfiguration() 메소드를 호출하여 jakarta.security.jacc.PolicyConfiguration 타입의 인스턴스(PC)를 리턴받는다.
6. Deployment 코드는 4번 과정에서 생성된 Role-to-Resource Permission들을 PC의 다양한 메소드를 사용하여 PC에 추가한다.
7. 모든 Permission을 PC에 추가하고 나면, PC의 commit() 메소드를 호출한다.

서블릿과 EJB 모듈이 undeploy될 때 PC.delete() 메소드가 호출된다. 이는 해당 서블릿 모듈과 EJB 모듈에 대한 Permission을 제거한다.



Policy 설정 과정에 대한 자세한 설명은 JACC 스펙을 참고한다.

6.2.3. Policy 결정 및 집행 규약

Policy 결정 및 집행 규약은 이름에서도 알 수 있듯이 EJB와 서블릿에서 접근 권한과 관련된 결정을 어떻게 내리고 집행할 것인지 기술하고 있다. 이는 앞서 설명한 2가지 세부 규약이 모두 충족된 이후에 적용된다.

Policy 결정 및 집행은 다음과 같은 과정으로 진행된다. 본 절에서는 서블릿을 예로 들어 설명하지만, EJB의 경우에도 아래의 설명과 동일하다.

1. 해당 서블릿 페이지에 대한 요청이 들어온다.
2. 서블릿 컨테이너는 jakarta.security.jacc.PolicyContext 클래스를 사용하여 JACC Context 정보를 설정한다.
3. 서블릿 컨테이너는 2가지 JACC Web Permission(jakarta.security.jacc 패키지에 정의되어 있다)을 생성한다. 이 Permission 인스턴스는 현재 요청한 서블릿에 설정된 Permission을 나타낸다.
4. 서블릿 컨테이너는 서블릿 요청자가 위의 2가지 Permission을 가졌는지 알아보기 위해 JACC Provider에 Query를 던진다. Query를 해석하는 데는 여러 가지 방법이 있다.

예를 들어 Policy.implies() 메소드로 확인할 수 있다. 이때 파라미터로 요청자의 Principal로 초기화된 java.security.ProtectionDomain에서 체크하는 모든 Permission(들)이 사용된다.

5. JACC Provider는 2번 과정에 설정된 context 정보, 3번에서 생성한 Permission 인스턴스, 요청자의 Principal(s), Principal-to-Role 매핑, Role-to-Resource 매핑 등을 모두 사용하여, 현재 요청자가 서블릿 페이지에 접근 권한을 가졌는지를 판단한다.
6. 권한 체크 결과가 양수값이라면 서블릿 컨테이너는 요청자가 서블릿 페이지에 접근하도록 허락한다. 그렇지 않다면, 권한 체크에 실패했다는 에러 페이지가 나타난다.



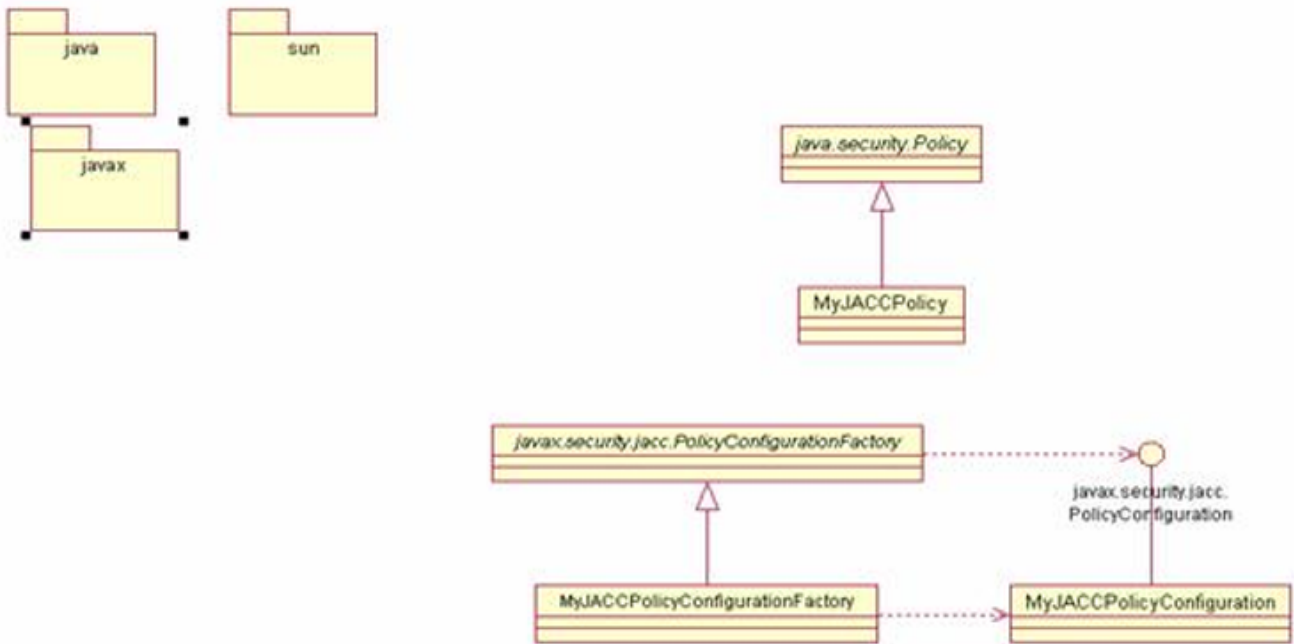
Policy 결정 및 집행 규약에 대한 자세한 설명은 JACC 스펙을 참고한다.

6.3. JACC Provider 개발

본 절에서는 Custom JACC Provider를 개발하는 방법과 JACC Provider 개발과 관련된 몇 가지 지침을 설명한다.

6.3.1. JACC Provider 구현

다음은 Custom JACC Provider를 구현하기 위해 필요한 클래스들과 그들 간의 관계를 단순하게 표현한 클래스 다이어그램이다. 이 다이어그램에서는 클래스 이름을 "MYJACC"로 시작했지만, 어떤 이름이 와도 상관없다.



JACC Provider 클래스

JACC Provider를 구현할 때 필요한 클래스는 다음과 같다.

- **java.security.Policy**

JACC Provider의 핵심은 java.security.Policy의 서브 클래스를 작성하는 것이다. 이 클래스는 JACC에서 실제 권한을 체크할 때 사용된다.

java.security.Policy의 서브 클래스를 작성하기 위해 java.security.Policy를 상속하는 새로운 클래스를 정의한다(이를 MyJACCPolicy라고 하자). 그리고 나서 [Policy 결정 및 집행 규약](#)에서 설명한 Policy 결정 및 집행 규약을 구현하기 위해 implies() 메소드를 재정의한다. 구현할 때 권한 체크 질의를 해석하기 위해 PolicyConfiguration 인터페이스를 통해 추가된 Permission과 Principal-to-Role 매핑을 고려해야 한다.

MyJACCPolicy는 파라미터가 없는 public 생성자를 제공해서 Jakarta EE 서버가 인스턴스를 쉽게 생성할 수 있게 해야 한다.

- **jakarta.security.jacc.PolicyConfigurationFactory**

Policy 설정 규약을 구현하기 위해 애플리케이션 서버는 Permission 인스턴스를 JACC java.security.Policy에 추가할 수 있어야 한다. 이 작업은 jakarta.security.jacc.PolicyConfiguration 인터페이스를 통해 이루어진다. PolicyConfiguration 인스턴스를 생성하기 위해 추상 클래스인 jakarta.security.jacc.PolicyConfigurationFactory가 필요하다.

PolicyConfigurationFactory의 서브 클래스를 작성하기 위해 jakarta.security.jacc.PolicyConfigurationFactory를 상속하는 새로운 클래스를 정의한다(이를 MyJACCPolicyConfigurationFactory라 하자). 이 클래스는 getPolicyConfiguration()를 반드시 재정의해야 한다.

MyJACCPolicyConfigurationFactory 클래스는 파라미터가 없는 public 생성자를 제공해서 Jakarta EE 서버가 해당 클래스의 인스턴스를 쉽게 생성할 수 있도록 해야 한다.

- **jakarta.security.jacc.PolicyConfiguration**

Policy 설정 규약을 구현하기 위해 애플리케이션 서버는 Permission 인스턴스를 JACC `java.security.Policy`에 추가할 수 있어야 한다. 이 작업은 `jakarta.security.jacc.PolicyConfiguration` 인터페이스를 통해 이루어진다. 이 인터페이스의 인스턴스는 위에서 설명한 대로 `jakarta.security.jacc.PolicyConfigurationFactory`를 통해 생성된다.

JACC Provider는 `jakarta.security.jacc.PolicyConfiguration`을 구현한 클래스를 반드시 포함하고 있어야 한다(이 구현 클래스를 `MyJACCPolicyConfiguration`라고 하자).

Custom `jakarta.security.jacc.PolicyConfigurationFactory` 클래스인

`MyJACCPolicyConfigurationFactory`는 항상 `MyJACCPolicyConfiguration` 인스턴스를 리턴해야 한다.



본 절에서는 각 클래스 구현 방법에 대해 대략적으로 설명했다. 구현하는 방법에 대한 자세한 설명은 JACC 스펙의 Policy 결정 및 집행 규약과 각 클래스에 대한 Jakarta EE Javadoc을 확인한다.

6.3.2. JACC Provider 패키징

일반적으로 JACC Provider와 Provider가 참조하는 클래스를 JAR로 묶어, 해당 애플리케이션 서버에 위치시킨다 (이를 "`MYJACCPProvider.jar`"라고 하자).

애플리케이션 서버의 Path에 JACC Provider JAR 파일의 Path를 포함시키고 필요한 경우에는 특정 시스템 속성값을 설정한다. 이 과정은 애플리케이션 서버마다 조금씩 달라진다. JEUS에서의 설정 방법은 [JEUS 보안 시스템과 JACC Provider 통합](#)에서 설명한다.

6.3.3. Default JACC Provider

JEUS 보안 시스템은 매우 간단한 디폴트 JACC Provider를 제공하고 있다. 일반적으로 이 디폴트 Provider를 JACC 스펙을 테스트 이외의 용도로 사용하는 것을 권장하지 않는다. 대신 이전 장에서 설명한 Default Authorization Provider를 사용할 것을 권장한다.

그러나 디폴트 이외의 JACC Provider를 사용하려면 상용 제품은 제공되고 있지 않으므로 자신만의 JACC JAR Archive를 직접 만들고 Archive 내에 JACC Provider 파일을 포함시키도록 한다.

6.4. JEUS 보안 시스템과 JACC Provider 통합

본 절에서는 JEUS 보안 시스템과 디폴트 JACC Provider를 통합하는 방법에 대해 설명한다.

JEUS 보안 시스템과 JACC Provider를 통합하기 위한 과정은 다음과 같다.

1. Principal-to-Role 매핑을 구현한다.

JACC 인터페이스는 Principal-to-Role 매핑에 대해 어떤 것도 언급하고 있지 않고 단지 Role-to-Resource 매핑만 정의하고 있다. Principal-to-Role 매핑을 위해 JEUS만의 별도의 인터페이스가 필요하다. 이를 위해 JEUS는 `jeus.security.impl.aznrep.JACCPrincipalRoleMapper`라는 인터페이스를 제공한다. 이 인터페이스는 구현해야 할 단 하나의 메소드인 `addPrincipalRoleMapping(PermissionMap map, String`

policyId)을 포함하고 있다. 이 메소드는 Principal-to-Role 매핑을 policyId로 대표되는 PolicyConfiguration에 추가한다.



Principal-to-Role 매핑은 Jakarta EE에서 애플리케이션 범위를 가진다는 점에 주의한다. 동일 Jakarta EE 애플리케이션에 속한 모든 Principal-to-Role 매핑은 하나의 Map에 병합되기 때문이다. PermissionMap 클래스와 PermissionMap 인스턴스를 서로 병합하는데 사용되는 add() 메소드에 대한 자세한 정보는 PermissionMap 클래스에 대한 API 문서를 참고한다.

JACCPrincipalRoleMapper 인터페이스를 구현한 클래스는 파라미터가 없는 public 생성자를 제공해야 한다. 그리고 반드시 JACC Provider JAR 내에 추가되어야 한다. JACCPrincipalRoleMapper 인터페이스에 대한 자세한 정보는 [참고 자료](#)와 Javadoc을 확인한다.

JACCPrincipalRoleMapper가 Principal-to-Role 매핑을 생성하는 과정은 다음과 같다.

- jeus.security.jacc.principalRoleMapper 시스템 속성으로 jeus.security.jacc.principalRoleMapper를 구현한 클래스명을 설정한다.
- 설정이 완료되면 jeus.security.impl.aznrep.JACCAuthorizationRepositoryService를 구현한 클래스가 이 속성값을 읽어 Class.forName(mapperClassname).newInstance() 메소드로 해당 인스턴스를 생성한다.
- 생성된 인스턴스의 addPrincipalRoleMapping() 메소드를 호출하여 JEUS DD 파일에 명시된 Principal-to-Role 매핑을 생성해서 추가한다.

2. 보안 설정 파일을 설정한다.

JEUS 보안 시스템은 2개의 어댑터 클래스를 사용하여 JEUS native authorization API와 JACC authorization API를 연결한다.

2개의 어댑터 클래스와 각각의 역할은 다음과 같다.

- jeus.security.impl.azn.JACCAuthorizationService

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

- jeus.security.impl.aznrep.JACCAuthorizationRepositoryService

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

JACC를 작동시키기 위해서는 이 2가지 보안 서비스가 security-domains.xml의 도메인 서비스 정의에 설정되어 있어야 한다.

JACC 보안 설정 파일 설정 : <security-domains.xml>

```
<?xml version="1.0"?>
<security-domains>
  ...
  <security-domain>
    <name>JACC_DOMAIN</name>
```

```

    <authorization>
      <jacc-service/>
    </authorization>
  </security-domain>
  . . .
</security-domains>

```

3. 시스템 Path에 JACC Provider JAR Path를 추가한다.

시스템 Path에 JACC Provider JAR 파일을 추가하기 위해 다음의 디렉터리에 JAR 파일을 위치시킨다.

```
JEUS_HOME/lib/system
```

4. Java 시스템 속성을 설정한다.

JACC 규약과 JEUS는 애플리케이션 서버가 JACC Provider를 인식하도록 다음 3가지 Java 시스템 속성을 규정하고 있다.

- jakarta.security.jacc.policy.provider

JACC Provider를 나타내며, java.security.Policy를 구현한 클래스명이다.

- jakarta.security.jacc.PolicyConfigurationFactory.provider

PolicyConfiguration 인스턴스를 생성하고 로딩하는 PolicyConfigurationFactory를 구현한 클래스명이다.

- jeus.security.jacc.principalRoleMapper

jeus.security.impl.aznrep.JACCPrincipalRoleMapper 인터페이스를 구현한 클래스명으로 JEUS DD 파일로부터 Principal-to-Role 매핑을 생성한다.

위의 3가지 시스템 속성은 모든 서버에 대해서 domain.xml에 <jvm-option>으로 설정되어야 한다.

JACC에 대한 Java 시스템 속성 설정 : <domain.xml>

```

<?xml version="1.0"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <servers>
    <server>
      <name>server1</name>
      <!-- server JVM option -->

      <jvm-config>
        . . .
        <jvm-option>
          -Djakarta.security.jacc.policy.provider=
            myprovider.MyJACCPolicy
        </jvm-option>
        <jvm-option>
          -Djakarta.security.jacc.PolicyConfigurationFactory.provider=
            myprovider.MyJACCPolicyConfigurationFactory
        </jvm-option>
      </jvm-config>
    </server>
  </servers>
</domain>

```



```

        <jvm-option>
            -Djeus.security.jacc.principalRoleMapper=
              myprovider.MyJACCPrincipalToRoleMapper
        </jvm-option>
    </jvm-config>
    . . .

```

디폴트 JACC Provider 클래스

이전에 언급한 대로 디폴트 JACC provider 클래스명은 다음과 같다.

구분	클래스명
Policy	jeus.security.impl.jacc.JACCPolicyWrapper
PolicyConfigurationFactory	jeus.security.impl.jacc.JACCPolicyConfigurationFactoryImpl
JACCPrincipalRoleMapper	jeus.security.impl.jacc.JACCDefaultPrincipalRoleMapper

이 클래스들은 모두 다음의 위치에 패키징되어 있다.

```
JEUS_HOME/lib/system/jeus.jar
```

7. JAAS 사용

본 장에서는 JAAS 개념에 대해서 살펴보고, JEUS 보안 시스템에서 SunOne Directory Server와 연동하는 방법을 간단히 설명한다.

7.1. 개요

Java 인증 및 승인 서비스 JAAS(Java Authentication and Authorization Service)는 Java 버전의 표준 PAM(Pluggable Authentication Module)을 구현하고 사용자 기반 인증을 지원하여 액세스 제어 인증 및 수행을 위한 표준 기반의 서비스 패키지이다.

JAAS는 Java 2 SDK, Standard Edition(J2SDK) 3의 옵션 패키지(확장 기능)였다. JAAS는 J2SDK 4 이후에 통합되어 있다.

JAAS 배경에는 다음의 2가지 기본 목적이 있다.

- 사용자를 **인증**한다.

코드가 애플리케이션, 애플릿, Bean 또는 서블릿인지에 관계없이 Java 코드를 실행 중인 사용자를 신뢰하도록 안전한 방법으로 확인한다.

- 사용자를 **승인**한다.

동작의 실행에 필요한 액세스 제어권(액세스권)을 사용자가 보관 유지하고 있는 것을 확인한다.

지금까지 Java 2는 코드 소스 베이스의 액세스 제어(코드의 출처(소) 및 서명자에 근거하는 액세스 제어)를 제공해 왔다. 그러나 코드의 실행자에 근거하는 액세스 제어를 추가 실행하는 기능은 부족하여 JAAS가 Java 2 보안 아키텍처를 확장하여 지원하도록 한다.

JAAS 인증은 **플러그인 가능** 방식에서 실행된다. 이 때문에 애플리케이션은 기반이 되는 인증 기술로부터 독립적으로 기능한다. 애플리케이션 내에서는 신규 또는 갱신된 인증 기술을 플러그인으로서 사용할 수 있고, 애플리케이션을 변경할 필요는 없다.

애플리케이션은 LoginContext Object를 인스턴스화하는 것으로 인증 프로세스를 유효하게 한다. LoginContext Object는 Configuration을 참조하여 사용하는 인증 테크놀로지 또는 LoginModule을 결정한다. 일반적인 LoginModule은 사용자명 및 패스워드의 입력을 요청해 입력된 것을 검증하고 음성이나 지문의 독해 및 Object로 검증을 실행할 수 있는 것도 있다.

코드를 실행하는 사용자가 인증되면 JAAS 승인 컴퍼넌트는 코어 Java 액세스 제어 모델과 연동해 자원의 액세스를 보호한다. 액세스 제어의 결정이 코드 위치 및 코드 서명자(CodeSource)에만 기초를 두는 J2SDK 3과는 달리, J2SDK 4부터는 액세스 제어의 결정은 실행 코드의 소스 및 코드를 실행하는 Subject Object로 나타낼 수 있는 사용자 또는 서비스에 근거하고 있다. 인증에 성공했을 경우 로그인 모듈은 관련하는 Principal 및 자격을 사용해 Subject를 갱신한다.



JAAS 스펙에 대한 자세한 정보는 [JAAS 튜토리얼 문서](#)를 참고한다.

기본적으로 JAAS 인증 매커니즘이 적용된 LDAP와 데이터베이스, JEUS 보안 시스템 간의 연동을 위해서

LoginModule을 확장 구현하고, JEUS 보안 시스템에서는 로그인과 승인에 관련한 서비스를 확장 제공한다.

본 장에서는 JEUS 보안 시스템에 JAAS 관련 코어 클래스 및 인터페이스를 적용해서 LDAP와 데이터베이스 연동 예제에 대해 설명한다.

- LDAP LoginModule 연동 예
- DBRealm LoginModule 연동 예

7.2. JEUS-LDAP 연동 위한 LoginModule 구현

javax.security.auth.login.LoginContext 인터페이스 클래스는 피인증자의 인증에 사용되는 기본적인 메소드를 제공하는 클래스이다. 이 클래스를 사용하면 애플리케이션에 로그인하는 것으로 기반이 되는 인증 테크놀로지에 의존하지 않고 특정 타입의 인증 타입을 제공하는 애플리케이션을 개발할 수 있다.

LDAP와 연동을 위해서 LoginModule을 확장하여 JAAS 인증 매커니즘을 지원하도록 한다. LoginModule 인터페이스를 상속한 LdapLoginModule을 통해서 인증 매커니즘을 수행하고 있다.

LdapLoginModule을 JEUS와의 보안 시스템과의 연동을 고려하여 다음과 같이 확장 구현한다.

jeus.security.impl.login.LdapLoginModule

```
package jeus.security.impl.login;

import jeus.security.base.Domain;
import jeus.security.resource.Password;
import jeus.security.resource.PrincipalImpl;
import jeus.security.resource.RolePrincipalImpl;
import jeus.util.logging.JeusLogger;

import javax.security.auth.Subject;
import javax.security.auth.callback.*;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import java.security.Principal;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;

public class LdapLoginModule implements LoginModule {
    protected static final JeusLogger logger = (JeusLogger)
JeusLogger.getLogger("jeus.security.loginmodule");

    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;

    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    private String username;
    private String password;
    private String domain;
```

```

private Principal userPrincipal;
private Password userCredential;

private SunOneLdapAuthenticator authenticator;

private static final String INITIAL_CONTEXT_FACTORY = "initialContextFactory"; // optional
private static final String PROVIDER_URL = "providerURL";
private static final String CONNECTION_USERNAME = "connectionUsername";
private static final String CONNECTION_PASSWORD = "connectionPassword";
private static final String USER_BASE = "userBase";
private static final String USER_SEARCH_MAPPING = "userSearchMapping";
private static final String USER_PASSWORD_ATTR = "userPasswordAttr";
private static final String USER_ROLE_ATTR = "userRoleAttr";
private static final String ROLE_BASE = "roleBase";
private static final String ROLE_NAME_ATTR = "roleNameAttr";
private static final String ROLE_SEARCH_MAPPING = "roleSearchMapping";

private final String SUN_JDK_LDAP_CONTEXT_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";

public void initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState, Map
options) {

    this.subject = subject;
    this.callbackHandler = callbackHandler;
    this.domain = Domain.SYSTEM_DOMAIN_NAME;

    authenticator = new SunOneLdapAuthenticator();
    initAuthenticator(authenticator, options);
}

private void initAuthenticator(SunOneLdapAuthenticator authenticator, Map options) {
    // INITIAL_CONTEXT_FACTORY
    String value = (String) options.get(INITIAL_CONTEXT_FACTORY);
    if (value == null) {
        authenticator.setContextFactory(SUN_JDK_LDAP_CONTEXT_FACTORY);
    } else {
        authenticator.setContextFactory(value);
    }

    authenticator.setProviderUrl((String) options.get(PROVIDER_URL));
    authenticator.setConnectionUsername(
        (String) options.get(CONNECTION_USERNAME));
    authenticator.setConnectionPassword(
        (String) options.get(CONNECTION_PASSWORD));

    value = (String) options.get(USER_BASE);
    if (value != null) {
        authenticator.setUserBase(value);
    } else {
        String msg = "LoginMoulde initialization failed. " + "userBase option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }

    authenticator.setUserPasswordAttr((String) options.get(USER_PASSWORD_ATTR));

    value = (String) options.get(USER_SEARCH_MAPPING);
    if (value != null) {
        authenticator.setUserSearchMapping(new MessageFormat(value));
    }
}

```

```

    } else {
        String msg = "LoginMoulde initialization failed. " + "userSearchMapping option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }

    authenticator.setUserRoleAttr((String) options.get(USER_ROLE_ATTR));

    value = (String) options.get(ROLE_BASE);
    if (value != null) {
        authenticator.setRoleBase(value);
    } else {
        String msg = "LoginMoulde initialization failed. " + "roleBase option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }

    value = (String) options.get(ROLE_NAME_ATTR);
    if (value != null) {
        authenticator.setRoleNameAttr(value);
    } else {
        String msg = "LoginMoulde initialization failed. " + "roleNameAttr option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }

    value = (String) options.get(ROLE_SEARCH_MAPPING);
    if (value != null) {
        authenticator.setRoleSearchMapping(new MessageFormat(value));
    } else {
        String msg = "LoginMoulde initialization failed. " + "roleSearchMapping option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }
}

public boolean commit() throws LoginException {
    if (succeeded == false) {
        return false;
    } else {
        userPrincipal = new PrincipalImpl(username);
        if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);

        ArrayList roles = authenticator.getRoles();
        for (Iterator i = roles.iterator(); i.hasNext(); ) {
            String roleName = (String) i.next();
            logger.fine("Adding role to subject : username = " + username + ", roleName = " +
roleName);
            subject.getPrincipals().add(new RolePrincipalImpl(roleName));
        }

        userCredential = new Password(password);
        subject.getPrivateCredentials().add(userCredential);

        username = null;
        password = null;
        domain = null;
        commitSucceeded = true;
    }
}

```

```

        return true;
    }
}

public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        succeeded = false;
        username = null;
        password = null;
        domain = null;
        userPrincipal = null;
        userCredential = null;
    } else {
        logout();
    }
    return true;
}

public boolean logout() throws LoginException {
    subject.getPrincipals().remove(userPrincipal);
    subject.getPrivateCredentials().remove(userCredential);
    succeeded = false;
    succeeded = commitSucceeded;
    username = null;
    password = null;
    domain = null;
    userPrincipal = null;
    userCredential = null;
    return true;
}

public boolean login() throws LoginException {
    Callback[] callbacks = null;

    // prompt for a user name and password
    if (callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
                                   "to garner authentication information from the user");

    callbacks = new Callback[3];
    callbacks[0] = new NameCallback("user name: ");
    callbacks[1] = new PasswordCallback("password: ", false);
    callbacks[2] = new TextInputCallback("domain: ");
    try {
        callbackHandler.handle(callbacks);
        username = ((NameCallback) callbacks[0]).getName();
        char[] tmpPassword = ((PasswordCallback) callbacks[1]).getPassword();
        if (tmpPassword == null) {
            tmpPassword = new char[0];
        }
        password = new String(tmpPassword);
        ((PasswordCallback) callbacks[1]).clearPassword();
        domain = ((TextInputCallback) callbacks[2]).getText();

        if (!authenticator.authenticate(username, password)) {
            throw new LoginException("LDAP authentication failed.");
        }
    }
}

```

```

        succeeded = true;
    } catch (UnsupportedCallbackException uce) {
        uce.printStackTrace();
        LoginException le = new LoginException(
            "Error: " + uce.getCallback().toString() +
            " not available to garner authentication information " +
            "from the user");
        le.initCause(uce);
        throw le;
    } catch (Exception e) {
        e.printStackTrace(); // todo. logging
        if (e instanceof LoginException) {
            throw (LoginException) e;
        } else {
            LoginException le = new LoginException(e.toString());
            le.initCause(e);
            throw le;
        }
    }
    return succeeded;
}
}

```



런타임에 추가되는 사용자와 역할 정보가 JEUS 보안 시스템에 적용이 되어야 하므로 RolePrincipalImpl 정보가 JEUS의 RolePrincipalImpl 타입으로 변환되어야 한다.

7.3. LDAP JAAS LoginModule 서비스 설정

JEUS 보안 시스템에서 특정 도메인에 JAAS LoginModule을 적용하여 로그인 서비스를 제공하고자 한다면 도메인 보안 서비스를 설정할 때 <jaas-login-config>를 설정한다. 자세한 사항은 [보안 서비스 설정](#)의 'authentication' 항목을 참고한다.

DEFAULT_APPLICATION_DOMAIN에 LDAP LoginModule 보안 서비스를 제공하기 위해서 로그인 서비스와 승인 서비스에 대해서 다음과 같이 설정한다.

LDAP JAAS LoginModule 서비스 설정 : <security-domains.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<security-domains xmlns="http://www.tmaxsoft.com/xml/ns/jeus" version="9.0">
    . . .
    <security-domain>
        <name>DEFAULT_APPLICATION_DOMAIN</name>
        <authentication>
            <jaas-login-config>
                <login-module>
                    <login-module-classname>
                        jeus.security.impl.login.LdapLoginModule
                    </login-module-classname>
                    <control-flag>required</control-flag>
                    <option>
                        <name>initialContextFactory</name>
                        <value>com.sun.jndi.ldap.LdapCtxFactory</value>
                    </option>
                </login-module>
            </jaas-login-config>
        </authentication>
    </security-domain>

```

```

        <option>
            <name>providerURL</name>
            <value>ldap://192.168.1.63:389</value>
        </option>
        <option>
            <name>connectionUsername</name>
            <value>cn=Directory Manager</value>
        </option>
        <option>
            <name>connectionPassword</name>
            <value>adminadmin</value>
        </option>
        <option>
            <name>userBase</name>
            <value>ou=People,dc=sample,dc=com</value>
        </option>
        <option>
            <name>userSearchMapping</name>
            <value>(uid={0})</value>
        </option>
        <option>
            <name>roleBase</name>
            <value>ou=Groups,dc=sample,dc=com</value>
        </option>
        <option>
            <name>roleNameAttr</name>
            <value>cn</value>
        </option>
        <option>
            <name>roleSearchMapping</name>
            <value>(uniqueMember={0})</value>
        </option>
    </login-module>
</jaas-login-config>
</authentication>
<authorization>
    <repository-service>
        <custom-repository>
            <classname>
jeus.security.impl.aznrep.CustomPolicyFileRealmAuthorizationRepositoryService
            </classname>
            <property>
                <name>PolicyClassName</name>
                <value>jeus.security.base.CustomJeusPolicy</value>
            </property>
            <property>
                <name>UserPrincipalClassName</name>
                <value>jeus.security.resource.PrincipalImpl</value>
            </property>
            <property>
                <name>RolePrincipalClassName</name>
                <value>jeus.security.resource.RolePrincipalImpl</value>
            </property>
        </custom-repository>
    </repository-service>
</authorization>
</security-domain>
. . .
</security-domains>

```


각 클래스에 대한 설명은 다음과 같다.

- jeus.security.impl.callback.JAASUsernamePasswordCallbackHandler

JEUS 보안 시스템에서 LoginModule에 인증 정보(username, password, etc..)를 취득하는 기본 매커니즘을 제공하는 CallbackHandler 클래스이다.

- jeus.security.impl.login.LdapLoginModule

JEUS 보안 시스템에서 옵션값으로 정의된 LDAP Attribute 값을 토대로 LoginModule을 지원하는 LoginModule 구현체 클래스이다.

- jeus.security.impl.aznrep.CustomPolicyFileRealmAuthorizationRepositoryService

JEUS 보안 시스템에서 사용자가 정의한 UserPrincipalClassName/RolePrincipalClassName 타입을 적용하여 Authorization 서비스를 제공하는 클래스이다.

- jeus.security.base.CustomJeusPolicy

JEUS 보안 시스템에서 런타임에 jeus-web-dd.xml이 존재하지 않고 LoginModule에서 Principal에 대한 RolePrincipalImpl이 정의하는 경우 Principal-to-Role 매핑이 되도록 지원하는 jeus.security.base.Policy를 확장 구현한 클래스이다.

설정이 완료되면 JEUS를 기동하고 DEFAULT_APPLICATION_DOMAIN에 deploy된 애플리케이션에 대한 로그인 서비스를 시작한다.

7.4. 데이터베이스 연동 위한 LoginModule 구현

LoginModule 인터페이스를 상속한 DBRealmLoginModule을 통해서 인증 메커니즘을 수행하고 있다. DBRealmLoginModule을 JEUS와의 보안 시스템과의 연동을 고려하여 다음과 같이 확장 구현한다.

jeus.security.impl.login.DBRealmLoginModule

```
package jeus.security.impl.login;

import jeus.security.base.Domain;
import jeus.security.base.ServiceException;
import jeus.security.resource.Password;
import jeus.security.resource.PrincipalImpl;
import jeus.security.resource.RolePrincipalImpl;
import jeus.util.logging.JeusLogger;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.security.auth.Subject;
import javax.security.auth.callback.*;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.sql.DataSource;
import java.security.Principal;
import java.sql.Connection;
import java.sql.PreparedStatement;
```

```

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

/**
 * User: choco
 * Date: 2016. 09. 13
 * Time: 오후 4:35:57
 */
public class DBRealmLoginModule implements LoginModule {
    protected static final JeusLogger logger = (JeusLogger)
JeusLogger.getLogger("jeus.security.login");
    protected String dsExportName;
    protected String principalsQuery = "select password from jeus_users where username=?";
    protected String rolesQuery = "select role from jeus_roles where username=?";

    private String username;
    private String password;
    private String domain;

    private Subject subject;
    private CallbackHandler callbackHandler;
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    protected Map options;
    private Principal userPrincipal;
    private Password userCredential;

    public void initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState, Map
options) {
        this.subject = subject;
        this.callbackHandler = callbackHandler;
        this.options = options;

        try {
            domain = Domain.getCurrentDomain().getName();
        } catch (ServiceException e) {
            domain = Domain.SYSTEM_DOMAIN_NAME;
        }

        dsExportName = (String) options.get("exportName");
        if (dsExportName == null) {
            String msg = "LoginMoulde initialization failed. " + "exportName option missing.";
            logger.warning(msg);
            throw new IllegalArgumentException(msg);
        }

        Object tmp = options.get("principalsQuery");
        if (tmp != null)
            principalsQuery = tmp.toString();
        tmp = options.get("rolesQuery");
        if (tmp != null)
            rolesQuery = tmp.toString();
        logger.debug("DBRealmLoginModule, export name : " + dsExportName);
        logger.debug("principalsQuery=" + principalsQuery);
    }

```

```

        logger.debug("rolesQuery=" + rolesQuery);
        logger.debug("initialize successfully");
    }

    public boolean login() throws LoginException {
        Callback[] callbacks = null;

        // prompt for a user name and password
        if (callbackHandler == null)
            throw new LoginException("Error: no CallbackHandler available " + "to garner
authentication information from the user");

        callbacks = new Callback[3];
        callbacks[0] = new NameCallback("user name: ");
        callbacks[1] = new PasswordCallback("password: ", false);
        callbacks[2] = new TextInputCallback("domain: ");
        try {
            callbackHandler.handle(callbacks);
            username = ((NameCallback) callbacks[0]).getName();
            char[] tmpPassword = ((PasswordCallback) callbacks[1]).getPassword();
            if (tmpPassword == null) {
                tmpPassword = new char[0];
            }
            password = new String(tmpPassword);
            ((PasswordCallback) callbacks[1]).clearPassword();
            domain = ((TextInputCallback) callbacks[2]).getText();

            String expectedPassword = getUsersPassword();
            if (validatePassword(password, expectedPassword) == false) {
                throw new LoginException("DBRealm authentication failed.");
            }
            succeeded = true;
        } catch (UnsupportedCallbackException uce) {
            uce.printStackTrace();
            LoginException le = new LoginException(
                "Error: " + uce.getCallback().toString() +
                " not available to garner authentication information " +
                "from the user");
            le.initCause(uce);
            throw le;
        } catch (Exception e) {
            e.printStackTrace();    // todo. logging
            if (e instanceof LoginException) {
                throw (LoginException) e;
            } else {
                LoginException le = new LoginException(e.toString());
                le.initCause(e);
                throw le;
            }
        }
        return succeeded;
    }

    private String getUsersPassword() throws LoginException {
        String password = null;
        Connection conn = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
    }

```

```

    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup(dsExportName);
        conn = ds.getConnection();
        ps = conn.prepareStatement(principalsQuery);
        ps.setString(1, username);
        rs = ps.executeQuery();
        if (rs.next() == false)
            throw new FailedLoginException("No matching username found");

        password = rs.getString(1);
    }
    catch (NamingException ex) {
        throw new LoginException(ex.toString(true));
    }
    catch (SQLException ex) {
        logger.debug("Query failed", ex);
        throw new LoginException(ex.toString());
    }
    finally {
        if (rs != null) {
            try {
                rs.close();
            }
            catch (SQLException e) {
            }
        }
        if (ps != null) {
            try {
                ps.close();
            }
            catch (SQLException e) {
            }
        }
        if (conn != null) {
            try {
                conn.close();
            }
            catch (SQLException ex) {
            }
        }
    }
    return password;
}

public boolean logout() throws LoginException {
    subject.getPrincipals().remove(userPrincipal);
    subject.getPrivateCredentials().remove(userCredential);
    succeeded = false;
    succeeded = commitSucceeded;
    username = null;
    password = null;
    domain = null;
    // trusted = false;
    userPrincipal = null;
    userCredential = null;
    return true;
}

```

```

public boolean commit() throws LoginException {
    if (succeeded == false) {
        return false;
    } else {
        userPrincipal = new PrincipalImpl(username);
        if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);

        ArrayList roles = getRoleSets();
        for (Iterator i = roles.iterator(); i.hasNext();) {
            String roleName = (String) i.next();
            logger.debug("Adding role to subject : username = " + username + ", roleName = " +
roleName);
            subject.getPrincipals().add(new RolePrincipalImpl(roleName));
        }

        userCredential = new Password(password);
        subject.getPrivateCredentials().add(userCredential);

        username = null;
        password = null;
        domain = null;
        commitSucceeded = true;
        return true;
    }
}

public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        succeeded = false;
        username = null;
        password = null;
        domain = null;
        userPrincipal = null;
        userCredential = null;
    } else {
        logout();
    }
    return true;
}

protected ArrayList getRoleSets() throws LoginException {
    Connection conn = null;
    HashMap setsMap = new HashMap();
    PreparedStatement ps = null;
    ResultSet rs = null;
    ArrayList roles = new ArrayList();
    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup(dsExportName);
        conn = ds.getConnection();
        ps = conn.prepareStatement(rolesQuery);
        try {
            ps.setString(1, username);
        }
        catch (ArrayIndexOutOfBoundsException ignore) {
        }
    }
}

```

```

        rs = ps.executeQuery();

        while (rs.next()) {
            String rolename = rs.getString(1);
            roles.add(rolename);
        }
    }
    catch (NamingException ex) {
        throw new LoginException(ex.toString(true));
    }
    catch (SQLException ex) {
        logger.debug("SQL failure", ex);
        throw new LoginException(ex.toString());
    }
    finally {
        if (rs != null) {
            try {
                rs.close();
            }
            catch (SQLException e) {
            }
        }
        if (ps != null) {
            try {
                ps.close();
            }
            catch (SQLException e) {
            }
        }
        if (conn != null) {
            try {
                conn.close();
            }
            catch (Exception ex) {
            }
        }
    }

    return roles;
}

protected boolean validatePassword(String inputPassword, String expectedPassword) {
    if (inputPassword == null || expectedPassword == null)
        return false;
    return inputPassword.equals(expectedPassword);
}
}

```



런타임에 추가되는 사용자와 역할 정보가 JEUS 보안 시스템에 적용이 되어야 하므로 RolePrincipalImpl 정보가 JEUS의 RolePrincipalImpl 타입으로 변환되어야 한다.

7.5. 데이터베이스 LoginModule 서비스 설정

JEUS 보안 시스템에서 특정 도메인에 JAAS LoginModule을 적용하여 로그인 서비스를 제공하려면 도메인 보안 서비스를 <jaas-login-config>에 설정한다. 자세한 사항은 [보안 서비스 설정](#)의 'authentication' 항목을 참고한다.

DEFAULT_APPLICATION_DOMAIN에 데이터베이스 LoginModule 보안 서비스를 제공하기 위해서 로그인 서비스와 승인 서비스에 대해서 다음과 같이 설정한다.

도메인 서비스 설정 : <security-domains.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<security-domains xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    . . .
    <security-domain>
        <name>DEFAULT_APPLICATION_DOMAIN</name>
        <authentication>
            <jaas-login-config>
                <login-module>
                    <login-module-classname>
                        jeus.security.impl.login.DBRealmLoginModule
                    </login-module-classname>
                    <control-flag>required</control-flag>
                    <option>
                        <name>exportName</name>
                        <value>dbrealmtest</value>
                    </option>
                    <option>
                        <name>principalsQuery</name>
                        <value>
                            select password from DEFAULT_APPLICATION_DOMAIN_Principals
                            where username=?
                        </value>
                    </option>
                    <option>
                        <name>rolesQuery</name>
                        <value>
                            select role from DEFAULT_APPLICATION_DOMAIN_roles
                            where username=?
                        </value>
                    </option>
                </login-module>
            </jaas-login-config>
        </authentication>
        <authorization>
            <repository-service>
                <custom-repository>
                    <classname>
                        jeus.security.impl.aznrep.
                        CustomPolicyFileRealmAuthorizationRepositoryService
                    </classname>
                    <property>
                        <name>PolicyClassName</name>
                        <value>jeus.security.base.CustomJeusPolicy</value>
                    </property>
                    <property>
                        <name>UserPrincipalClassName</name>
                        <value>jeus.security.resource.PrincipalImpl</value>
                    </property>
                    <property>
```

```

        <name>RolePrincipalClassName</name>
        <value>jeus.security.resource.RolePrincipalImpl</value>
    </property>
</custom-repository>
</repository-service>
</authorization>
<security-domain>
. . .
</security-domains>

```

다음은 각 클래스에 대한 설명이다.

- jeus.security.impl.login.DBRealmLoginModule

JEUS 보안 시스템에서 옵션값으로 데이터베이스의 exportname과 principal과 role에 대한 Query 값을 토대로 LoginModule을 지원하는 LoginModule 구현체 클래스이다.

다음은 옵션 항목에 대한 설명이다.

항목	설명
exportName	domain.xml에 정의된 데이터베이스의 export-name을 설정한다.
principalsQuery	Primary Key가 하나이고 principal에 대한 패스워드값을 얻어올 수 있는 Query를 정의한다.
rolesQuery	Primary Key가 하나이고 principal에 대한 role값을 얻어올 수 있는 Query를 정의한다.

- jeus.security.impl.aznrep.CustomPolicyFileRealmAuthorizationRepositoryService

JEUS 보안 시스템에서 사용자가 정의한 UserPrincipalClassName/RolePrincipalClassName 타입을 적용하여 Authorization 서비스를 제공하는 클래스이다.

- jeus.security.base.CustomJeusPolicy

JEUS 보안 시스템에서 런타임에 jeus-web-dd.xml descriptor가 존재하지 않고, LoginModule에서 principal에 대한 RolePrincipalImpl을 정의해야 하는 경우 Principal-to-Role 매핑이 되도록 지원하는 jeus.security.base.Policy를 확장 구현한 클래스이다.



설정이 완료되면 JEUS를 기동하고 DEFAULT_APPLICATION_DOMAIN에 deploy된 애플리케이션의 로그인 서비스를 시작한다.

Appendix A: 보안 이벤트 서비스

보안 이벤트 서비스에 대해 설명한다.

A.1. 개요

SPI 클래스와 디폴트 보안 서비스 구현 클래스로부터 EventHandlingService로 방출되는 표준 보안 이벤트(Security Event)에 대해 설명한다. 이 내용은 jeus.security.spi.EventHandlingService SPI를 구현하여 자신만의 이벤트 핸들링을 개발하는 데 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 context에 대한 key-value 쌍의 집합.
Emitted When? 이벤트가 발생하는 조건

보통 이벤트는 이벤트 소스와 같은 도메인에 있는 EventHandlingService들로 방출된다. 그러나, 예외로 security.install.successful과 security.uninstall.attempt 이벤트는 보안 시스템에 설정되어 있는 모든 도메인으로 방출된다.



jeus.security.base.Event 클래스와 jeus.security.spi.EventHandlingService 클래스에 대한 더 자세한 정보는 Javadoc을 참고한다.

A.2. 이벤트

다음은 표준 보안 이벤트의 목록이다.

security.validation.failed

Source Class	jeus.security.spi.SubjectValidationService
Event Type	security.validation.failed
Event Level	WARNING
Event Context	<ul style="list-style-type: none">◦ Key : "subject"◦ Value : 유효성 검증이 실패한 jeus.security.base.Subject
Emitted When	SubjectValidationService가 SecurityException을 발생할 때마다

security.authentication.failed

Source Class	jeus.security.spi.AuthenticationService
Event Type	security.authentication.failed
Event Level	WARNING
Event Context	<ul style="list-style-type: none">◦ Key : "subject"◦ Value : 유효성 검증이 실패한 jeus.security.base.Subject
Emitted When	Subject에 대한 사용자 인증이 실패할 때마다

security.authorization.failed

Source Class	jeus.security.spi.AuthorizationService
Event Type	security.authentication.failed
Event Level	WARNING
Event Context	<ul style="list-style-type: none">◦ 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

security.authentication.repository.subject.added

Source Class	jeus.security.spi.AuthenticationRepositoryService
Event Type	security.authentication.repository.subject.added
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none">◦ Key : "subject"◦ Value : 추가된 jeus.security.base.Subject
Emitted When	AuthenticationRepositoryService에 Subject가 성공적으로 추가될 때마다

security.authentication.repository.subject.removed

Source Class	jeus.security.spi.AuthenticationRepositoryService
--------------	---

Event Type	security.authentication.repository.subject.removed
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> ◦ Key : "subject" ◦ Value : 삭제된 jeus.security.base.Subject
Emitted When	AuthenticationRepositoryService로부터 Subject가 성공적으로 삭제될 때마다

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	<ul style="list-style-type: none"> ◦ Key : "name" ◦ Value : 성공적으로 삭제된 Subject
Emitted When	Subject가 AuthenticationRepositoryService로부터 성공적으로 삭제될 때마다

security.authorization.repository.policy.added

Source Class	jeus.security.spi.AuthorizationRepositoryService
Event Type	security.authorization.repository.policy.added
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> ◦ Key : "policy" ◦ Value : 추가된 jeus.security.base.Policy
Emitted When	AuthorizationRepositoryService에 새로운 Policy가 추가될 때마다

security.authorization.repository.policy.removed

Source Class	jeus.security.spi.AuthorizationRepositoryService
Event Type	security.authorization.repository.policy.removed
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> ◦ Key : "policy" ◦ Value : 삭제된 jeus.security.base.Policy
Emitted When	AuthorizationRepositoryService로부터 Policy 데이터가 삭제될 때마다

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	<ul style="list-style-type: none"> ◦ Key : "contextid" ◦ Value : 저장소로부터 삭제된 java.lang.String 타입의 Context ID
Emitted When	AuthorizationRepositoryService로부터 context id가 삭제될 때마다

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

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

Appendix B: JEUS Server Permissions

본 부록은 표준 Permission 리소스 이름과 리소스 액션을 설명한다.

B.1. 개요

표준 Permission 리소스 이름과 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.ResourcePermission이며, Context ID는 항상 "default"이다.

권한 체크는 언제나 리소스 이름과 리소스 액션의 조합으로 일어난다. 보통의 경우 리소스 이름은 접근할 타겟을 의미하고, 리소스 액션은 해당 타겟에 대해 무엇을 할 것인지를 의미한다. 만약 원하는 권한 설정이 제대로 된 것 같지 않으면 Master Server나 서버 로그를 확인하여 정의된 Permission을 확인하여 추가한다.

B.2. JEUS 시스템 리소스 이름

여기서는 JEUS 기본 보안 시스템에서 제공되는 중요한 리소스 이름에 대해서만 소개한다.

Resource name	설명
jeus.*	JEUS 시스템의 모든 리소스 이름에 접근 가능하다.
jeus.server.<server-name>.*	JEUS 시스템의 특정 서버의 모든 리소스 이름에 접근 가능하다. 디폴트 보안 시스템을 사용할 때 서버측에서 체크하는 Permission 목록이다. 리소스 액션에 따라 아래와 같이 나뉜다. <ul style="list-style-type: none">◦ boot : 해당 서버를 시작시킬 때◦ down : 해당 서버를 종료시킬 때◦ deploy : 해당 서버에 애플리케이션을 deploy할 때◦ ftp : 파일을 전송에 FTP를 사용할 때
jeus.server.<server-name>.app.<application-name>	JEUS 시스템의 특정 서버의 특정 애플리케이션에 대한 리소스 이름이다.
jeus.cluster.<cluster-name>.*	JEUS 시스템의 특정 클러스터의 모든 리소스 이름에 접근 가능하다.
jeus.domain.<domain-name>	JEUS 시스템의 동적 설정변경 권한에 대한 리소스 이름이다. 여기서의 도메인은 보안 도메인이 아닌 JEUS 시스템 도메인을 의미한다. 리소스 액션에 따라 아래와 같이 나뉜다. <ul style="list-style-type: none">◦ dynamicConfiguration : jeusadmin 등을 통해 도메인의 설정을 동적으로 변경할 때

Resource name	설명
jeus.jndi	<p>JEUS 시스템의 JNDI 동작 권한에 대한 리소스 이름이다.</p> <p>리소스 액션에 따라 아래와 같이 나뉜다.</p> <ul style="list-style-type: none"> ◦ lookup : JNDI를 이용하여 객체를 lookup하려 할 때 ◦ modify : bind/unbind/rename과 같이 JNDI Repository 객체를 추가/삭제/변경할 때 ◦ list : JNDI repository에 저장된 객체의 List를 가져올 때
jeus.node.<node-name>	<p>JEUS 시스템의 특정 노드에 대한 리소스 이름이다. 노드를 추가, 삭제하는 경우 사용된다. 노드에 대한 설명은 "JEUS Server 안내서"을 참고한다.</p> <p>리소스 액션에 따라 아래와 같이 나뉜다.</p> <ul style="list-style-type: none"> ◦ edit : 노드를 추가하거나 삭제할 때

B.3. jeusadmin 명령어 권한 설정

디폴트 보안 시스템을 기준으로 jeusadmin의 명령 단위로 권한 설정이 가능하다.

jeusadmin 명령 단위로 권한을 설정한 경우에는, 해당 명령어에 대한 권한 확인만 이루어지고 나머지 내부 권한에 대해서는 체크하지 않고 넘어간다. 명령어 권한의 리소스 이름은 명령어의 옵션과 관련짓게 되는데, server, servers, cluster, clusters, node 옵션에 대해서 앞서 설명한 리소스 이름을 갖는다. 이 외의 모든 옵션에 대해서는 jeus.domain.<domain-name> 리소스 이름을 갖는다.

명령어 권한의 리소스 액션은 명령어의 이름으로 정의된다. 명령어의 alias에 대해서는 제공되지 않으므로 help <command-name>에서 확인되는 실제 명령어 이름을 넣어 주도록 한다. 명령어에 대한 정보는 JEUS Reference 안내서의 "Part II. 콘솔 명령어와 툴"을 참고한다.

다음은 "user1"에게는 관리자 권한, "user2"에게는 "server2"에 deploy 명령을 수행할 수 있는 권한을 주는 예제이다. "user2"의 경우 "server2"에 대한 권한만 주려고 하므로 리소스 이름 jeus.server.server2.*를 주고, jeusadmin에서 **deploy** 명령의 실제 이름이 deploy-application이므로 이를 리소스 액션에 넣어준다.

보안 시스템 Policy 설정 : <policies.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <role-permissions>
      <role-permission>
        <principal>user1</principal>
        <role>adminRole</role>
      </role-permission>
      <role-permission>
        <principal>user2</principal>
        <role>server2DeployRole</role>
      </role-permission>
    </role-permissions>
    <resource-permissions>
      <context-id>default</context-id>
```

```
<resource-permission>
  <role>adminRole</role>
  <resource>jeus.*</resource>
  <actions>*</actions>
</resource-permission>
<resource-permission>
  <role>server2DeployRole</role>
  <resource>jeus.server.server2.*</resource>
  <actions>deploy-application</actions>
</resource-permission>
</resource-permissions>
</policy>
</policies>
```

참고 자료

- Java Authorization Contract for Containers Specification Version 1.0

JACC provider에 대한 정보를 제공한다.

- Jakarta EE 9 Specification

JEUS를 포함한 Jakarta EE 서버에 적용되는 기본적인 보안 아키텍처에 대한 정보를 제공한다.

- EJB 4.0 Specification

EJB 보안 모델에 대한 정보를 제공한다.

- Servlet 5.0 Specification

Servlet 보안 모델에 대한 정보를 제공한다.

- Javadoc for java.security, javax.security.auth, jakarta.security.jacc

보안과 관련된 기본적인 클래스에 대한 상세한 정보를 제공한다.

- Javadoc JEUS API

JEUS_HOME/docs/api/jeusapi/index.html

- XML Reference - 보안 서비스 domain.xml 설정

JEUS_HOME/docs/reference/schema/index.html