Red Hat JBoss Enterprise Application Platform 7.1

セキュリティーアーキテクチャー

Red Hat JBoss Enterprise Application Platform 7.1 向け

Last Updated: 2018-08-29

Red Hat JBoss Enterprise Application Platform 7.1 セキュリティーアーキテクチャー

Red Hat JBoss Enterprise Application Platform 7.1 向け

法律上の通知

概要

本書では、JBoss EAP 内のセキュリティー概念の概要と、これらの概念の実装するために存在するコンポーネントを重点に取り上げます。 what (何であるか) と why (その理由) に重点を置き、how (やり方) は重点に置かないため、特定の場合の設定方法 については別のドキュメントを参照してください。本書をお読みいただくと、JBoss EAP 内のセキュリティーに関するコンポーネントの概念や、これらのコンポーネントがどのように組み合わさっているかを深く理解することができます。

目次

第1章 一般的なセキュリティー概念の概要

JBoss EAP がどのようにセキュリティーに対処するかを深く掘り下げる前に、基本のセキュリティー概念を理解することが重要です。

1.1. 認証

認証とは、認証の対象を特定し、その識別情報の真正性を検証することを言います。ユーザー名とパスワードの組み合わせが最も一般的な認証メカニズムですが、共有キー、スマートカード、指紋なども認証に使用されます。Java Enterprise Edition の宣言的セキュリティーでは、正常な認証の結果はプリンシパルと呼ばれます。

1.2. 承認

承認は、アクセス権を指定したり、アクセスポリシーを定義したりする方法のことです。システムはメカニズムを実装してこれらのポリシーを使用し、要求側のリソースへのアクセスを許可または拒否できます。多くの場合で、ロールと呼ばれる要求側のアクセスが許可されるアクションまたは場所のセットと、プリンシパルが一致すると実装されます。

1.3. 実際の認証および承認

認証と承認は異なる概念ですが、多くの場合で関係しています。認証の処理が目的で作成された多くのモジュールは承認も処理し、承認の処理が目的で作成された多くのモジュールは認証も処理します。

例

アプリケーション **MyPersonalSoapbox** はメッセージを投稿および閲覧する機能を提供します。**Talk** ロールを持つプリンシパルはメッセージを投稿でき、投稿された他のメッセージを閲覧できます。ログインしていないユーザーは **Listen** ロールを持ち、投稿されたメッセージを閲覧できます。Suzy、Adam、および Bob がアプリケーションを使用します。Suzy と Bob はユーザー名とパスワードで認証できますが、Adam のユーザー名とパスワードはまだ指定されていません。Suzy は **Talk** ロールを持ちますが、Bob は何のロールも持たず、**Talk** や **Listen** も持っていません。Suzy が認証されると、Suzy はメッセージを投稿し、閲覧することができます。Adam が **MyPersonalSoapbox** を使用すると、ログインはできませんが、投稿されたメッセージを閲覧できます。Bob はログインしてもメッセージを投稿できず、投稿された他のメッセージも見れません。

Suzy に対しては認証と承認の両方が行われます。Adam は認証されませんが、**Listen** ロールで承認され、メッセージを閲覧できます。Bob は認証されますが、承認されず、ロールを持ちません。

1.4. 暗号化

暗号化とは、数学的なアルゴリズムを適用して機密情報をエンコードすることを言います。データは、エンコードされた形式に変換 (暗号化) され、セキュア化されます。データを再度読み取るには、エンコードされた形式を元の形式に変換 (復号化) する必要があります。暗号化は、ファイルやデータベースの簡単な文字列データに適用でき、通信ストリーム全体に送信されたデータでも適用できます。

暗号化の例には以下の場合が含まれます。

- LUKS を使用して Linux ファイルシステムディスクを暗号化できます。
- blowfish または AES アルゴリズムを使用して Postgres データベースに格納されたデータを暗号化できます。

- HTTPS プロトコルは、SSL/TLS (Secure Sockets Layer/Transport Layer Security) 経由ですべてのデータを暗号化してから転送元から転送先へ送信します。
- ユーザーが SSH プロトコル (Secure Shell) を使用してあるサーバーから別のサーバーに接続する場合、すべての通信は暗号化されたトンネルで送信されます。

1.5. SSL/TLS および証明書

SSL/TLS は、2 つのシステム間でのみ交換および認識される対称キーを使用して、このシステム間のネットワークトラフィックを暗号化します。SSL/TLS は対称キーをセキュアに交換するため、暗号化でキーペアを使う PKI (Public Key Infrastructure) を使用します。キーペアは、個別のキーでありながら一致する、パブリックキーとプライベートキーの 2 つの暗号キーで構成されます。パブリックキーは第三者と共有でき、データの暗号化に使用されます。プライベートキーは秘密のキーとして扱われ、パブリックキーを使用して暗号化されたデータの復号化に使用されます。

クライアントが対称キーを交換するためにセキュアな接続を要求すると、セキュアな通信の開始前にハンドシェイクフェーズが発生します。SSL/TLS ハンドシェイクの間、サーバーはパブリックキーを証明書としてクライアントに渡します。証明書には、サーバーの識別情報、その URL、サーバーのパブリックキー、および証明書を検証するデジタル署名が含まれます。クライアントは証明書を検証し、信頼できる証明書であるかを判断します。証明書を信頼できる場合、クライアントは SSL/TLS 接続の対称キーを生成し、サーバーのパブリックキーを使用して暗号化し、サーバーに返送します。サーバーはプライベートキーを使用して対称キーを復号化します。その後、この接続上の 2 つのマシン間で行われる通信は対称キーを使用して暗号化されます。

証明書には、自己署名証明書と認証局署名証明書の 2 つの証明書があります。自己署名証明書はプライベートキーを使用してその証明書自体を署名します。信頼チェーンに接続されていないため、署名は検証されません。認証局署名証明書は認証局 (CA) によって発行される証明書で、VeriSign、CAcert、RSACA などの CA によって署名されます。CA は証明書の保持者の信頼性を検証します。

自己署名証明書の生成は迅速かつ簡単で、管理に必要なインフラストラクチャーが少なくなりますが、第三者によって信頼性が確認されないため、クライアントによる信頼性の検証が難しくなる可能性があります。そのため、自己署名証明書の安全性は低くなります。認証局署名証明書の設定はより手間がかかりますが、クライアントによる信頼性の検証が容易になります。第三者によって証明書の信頼性が確認されるため、信頼チェーンが作成されます。

警告

Red Hat では、影響するすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSLv2、SSLv3、および TLSv1.0 を明示的に無効化することを推奨しています。

1.6. シングルサインオン

シングルサインオン (SSO) は、1 つのリソースに対して認証されたプリンシパルが暗黙的に他のリソースへのアクセスを承認できるようにします。異なるリソースのセットが SSO によってセキュア化された場合、ユーザーはセキュア化されたこれらのリソースのいずれかに初めてアクセスするときのみ認証される必要があります。認証に成功した後、ユーザーに関連するロールが保存され、関連する他のリソースすべての承認に使用されます。これにより、ユーザーは再認証なしで、承認された他のリソースにアクセスできます。

ユーザーがリソースからログアウトした場合や、リソースがプログラムでセッションを無効化した場

合、永続化された承認データはすべて削除され、プロセスが再度開始されます。リソースのセッションタイムアウトが発生した場合は、そのユーザーに関連する有効なリソースセッションが他にあれば SSO セッションは無効化されません。SSO は、web アプリケーションやデスクトップアプリケーションでの認証および承認に使用することができます。場合によっては、SSO 実装は認証と承認の両方と統合できます。

SSO 内では、システムの異なる概念や部分を示すために共通の用語がいくつか使用されます。

アイデンティティー管理

アイデンティティー管理 (IDM) とは、1 つ以上のシステムやドメイン全体でプリンシパルとそれらに関連する認証、承認、および特権を管理する概念のことです。同じ概念がアイデンティティーおよびアクセス管理 (IAM) と呼ばれることもあります。

アイデンティティープロバイダー

アイデンティティープロバイダー (IDP) とはエンドユーザーを認証し、そのユーザーのアイデンティティーを信頼できる方法で信頼できるパートナーに対してアサートする、権限のあるエンティティーです。

アイデンティティーストア

アイデンティティープロバイダーには、認証および承認プロセス中に、使用するユーザーの情報を取得するアイデンティティーストアが必要です。データベース、LDAP (Lightweight Directory Access Protocol)、プロパティーファイルなど、あらゆるリポジトリーのタイプをアイデンティティーストアとして使用できます。

サービスプロバイダー

サービスプロバイダー (SP) は、アイデンティティープロバイダーに依存してユーザーの電子クレデンシャル経由でユーザーに関する情報をアサートし、信頼できるユーザークレデンシャルのアサートを基にアクセス制御と伝播を管理します。

クラスター化および非クラスター化 SSO

非クラスターか SSO は、同じ仮想ホストでアプリケーションへの承認情報の共有を制限します。また、ホストの障害発生時の回復性がありません。クラスター化された SSO の場合、複数の仮想ホストのアプリケーション間でデータが共有されるため、障害時の回復性があります。さらに、クラスター化された SSO はロードバランサーからリクエストを受信できます。

1.6.1. サードバーティー SSO 実装

Kerberos

Kerberos はクライアントサーバーアプリケーションのネットワーク認証プロトコルです。秘密鍵暗号方式を使用して、セキュアでないネットワーク上でセキュアな認証を実現します。

Kerberos はチケットと呼ばれるセキュリティートークンを使用します。セキュアなサービスを使用するには、ユーザーはネットワークのサーバーで実行されるサービスであるチケット付与サービス (TGS: Ticket Granting Service) からチケットを取得する必要があります。チケットの取得後、同じネットワークで実行される別のサービスである認証サービス (AS) からサービスチケット (ST) を要求します。その後、ユーザーは ST を使用して希望のサービスに対して認証されます。TGS と AS は、キー配布センター (KDC) と呼ばれるエンクロージングサービス内で実行されます。

Kerberos は、クライアントサーバーデスクトップ環境での使用を目的としているため、通常は web アプリケーションやシンクライアント環境では使用されません。しかし、 デスクトップの認証に Kerberos システムを使用しているため、web アプリケーション用のシステムを作成せずに、既存のシ

ステムを再使用したい組織が多くあります。Kerberos はMicrosoft Active Directory では不可欠なもので、Red Hat Enterprise Linux 環境の多くで使用されています。

SPNEGO

SPNEGO (Simple and protected GSS_API negotiation mechanism) は、web アプリケーションで使用するために Kerberos ベースの SSO 環境を拡張するメカニズムを提供します。

web ブラウザーなどのクライアントコンピューター上のアプリケーションが web サーバーの保護されたページにアクセスしようとすると、サーバーは承認が必要であることを伝えます。アプリケーションは KDC から ST を要求します。アプリケーションはチケットを SPNEGO 用にフォーマットされたリクエストにラップし、ブラウザー経由で web アプリケーションに返信します。デプロイされた web アプリケーションを実行している web コンテナーはリクエストをアンパックし、チケットを認証します。認証が正常に行われると、アクセスできるようになります。

SPNEGO は、Red Hat Enterprise Linux 内の Kerberos サービスや、Microsoft Active Directory の不可欠な要素である Kerberos サーバーなど、すべてのタイプの Kerberos プロバイダーと動作します。

Microsoft Active Directory

Active Directory (AD) は Microsoft 社によって開発されたディレクトリーサービスで、Microsoft Windows ドメインでユーザーとコンピューターを認証します。これは Windows Server の一部として提供されます。ドメインを制御する Windows Server を実行しているコンピューターはドメインコントローラーと呼ばれます。Red Hat Enterprise Linux は Active Directory ドメインと統合でき、Red Hat Identity Management、Red Hat JBoss Enterprise Application Platform、およびその他の Red Hat 製品とも統合できます。

Active Directory は、共に動作する 3 つのコアテクノロジーに依存します。

1. LDAP: ユーザー、コンピューター、パスワード、およびその他のリソースに関する情報を保存します。
2. Kerberos: ネットワーク上でセキュアな認証を提供します。
3. ドメインネームサービス (DNS): ネットワーク上のコンピューターおよびその他のデバイスにおける IP アドレスとホスト名との間のマッピングを提供します。

1.6.2. クレームベースのアイデンティティー

SSO を実装する方法の 1 つがクレームベースのアイデンティティーシステムを使用することです。クレームベースのアイデンティティーシステムでは、システムはアイデンティティー情報を渡すことができますが、その情報をクレームと発行者 (またはオーソリティー) の 2 つに抽象化します。クレームは、ユーザー、グループ、アプリケーション、組織などの 1 つのサブジェクトが他のサブジェクトに関して作成するステートメントです。1 つまたは複数のクレームは、1 つまたは複数のトークンにパッケージ化され、プロバイダーによって発行されます。クレームベースのアイデンティティーでは、個別のセキュアなリソースがユーザーに関する情報をすべて認識しなくても SSO を実装できます。

セキュリティートークンサービス

セキュリティートークンサービス (STS) は、セキュアなアプリケーション、web サービス、または EJB に対してユーザーを認証および承認するときに使用するセキュリティートークンをクライアントに発行する認証サービスです。STS でセキュア化されたアプリケーションに対して認証を試みるクライアント (サービスプロバイダー) は集中管理された STS オーセンティケーターにリダイレクトされ、トークンが発行されます。認証に成功すると、そのクライアントはトークンと元のリクエストを提供してサービスプロバイダーへ再度接続しようとします。そのサービスプロバイダーはクライアントのトークンを STS で検証し、処理を継続します。クライアントは STS に接続する他の web サービスや EJB に対

して同じトークンを再使用できます。セキュリティートークンを発行、キャンセル、更新、および検証でき、セキュリティートークンのリクエストおよび応答メッセージの形式を指定する集中管理された STS の概念は、**WS-Trust** と呼ばれます。

ブラウザーベースの SSO

ブラウザーベースの SSO では、サービスプロバイダーと呼ばれる 1 つ以上の web アプリケーションがハブアンドスポーク型アーキテクチャーの集中管理されたアイデンティティープロバイダーに接続します。IDP は、SAML トークンのクレームステートメントをサービスプロバイダー (スポーク) に発行し、アイデンティティーおよびロール情報の中心のソース (ハブ) として動作します。リクエストは、ユーザーがサービスプロバイダーへのアクセスを試みるときや、ユーザーが直接アイデンティティープロバイダーでの認証を試みるときに発行されます。これらはそれぞれ SP 開始 (SP-initiated) フローおよび IDP 開始 (IDP-initiated) フローと呼ばれ、どちらも同じクレームステートメントが発行されます。

SAML

SAML (Security Assertion Markup Language) は、通常はアイデンティティープロバイダーとサービスプロバイダーである 2 者が認証および承認情報を交換できるようにするデータ形式です。SAML トークンは、STS または IDP によって発行されるトークンの種類で、SSO の有効化に使用できます。SAML によってセキュア化されるリソースである SAML サービスプロバイダーは、STS または IDP の種類の 1 つである SAML アイデンティティープロバイダーにユーザーをリダイレクトし、そのユーザーを認証および承認する前に有効な SAML トークンを取得します。

デスクトップベースの SSO

デスクトップベースの SSO は、サービスプロバイダーとデスクトップドメイン (Active Directory または Kerberos など) がプリンシパルを共有できるようにします。これにより、ユーザーはドメインクレデンシャルを使用してコンピューターにログインでき、サービスプロバイダーは認証中にそのプリンシパルを再使用できるため、再認証や SSO の提供が必要ありません。

1.7. LDAP

LDAP (Lightweight Directory Access Protocol) はネットワーク全体でディレクトリー情報を格納および分散するプロトコルです。このディレクトリー情報には、ユーザー、ハードウェアデバイス、アクセスロール、制限に関する情報などが含まれます。

LDAP では、識別名 (DN) によってディレクトリーのオブジェクトが一意に識別されます。各識別名には、他のオブジェクトと区別するための一意名と場所が必要で、これには属性と値のペア (AVP) を使用します。AVP は、コモンネームや組織単位などの情報を定義します。LDAP に必要となる一意な文字列は、これらの値の組み合わせになります。

LDAP の一般的な実装には、Red Hat Directory Server、OpenLDAP, Active Directory、IBM Tivoli Directory Server、Oracle Internet Directory、および 389 Directory Server が含まれます。

第2章 初期状態の RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 7.1 によるセキュリティーの対処

JBoss EAP 7.1 には、セキュリティーに関する 3 つのコンポーネントが同梱されています。

- JBoss EAP 7.1 で導入された Elytron サブシステム
- Core Management Authentication
- The Security サブシステム

これらのコンポーネントは、一般的なセキュリティー概念の概要で説明した一般的なセキュリティー概念を基にしていますが、実装に JBoss EAP 固有の概念も取り入れています。

2.1. コアサービス、サブシステム、およびプロファイル

JBoss EAP は、モジュラークラスローディングの概念を基にして構築されています。JBoss EAP によって提供される各 API およびサービスはモジュールとして実装され、必要に応じてロードおよびアンロードされます。コアサービスは、サーバーの起動時に常にロードされるサービスで、追加のサブシステムを起動する前に実行する必要があります。

サブシステムは、拡張によってコアサーバーに追加される性能のことです。たとえば、異なるサブシステムによってサーブレットの処理、EJB コンテナーの管理、および JTA サポートの提供が行われます。

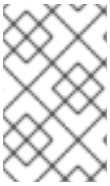
プロファイルはサブシステムの名前付きリストで、各サブシステムの設定の詳細と提供されます。サブシステムの数が多いプロファイルを使用すると、サーバーの性能も多くなります。プロファイルのサブシステムをより限定して少なくすると、性能も少なくなりますが、フットプリントが小さくなります。デフォルトでは、JBoss EAP には **default**、**full**、**ha**、**full-ha** などの事前定義されたプロファイルが複数含まれています。これらのプロファイルでは、管理インターフェースと関連するセキュリティーレルムはコアサービスとしてロードされます。

2.2. 管理インターフェース

JBoss EAP では、主に管理コンソールと管理 CLI の 2 つの管理インターフェースを使用して設定の指定や編集を行います。これらのインターフェースはいずれも JBoss EAP のコア管理の機能を公開し、同じコア管理システムにアクセスする方法を 2 つ提供します。

管理コンソールは、JBoss EAP の web ベースの管理ツールです。サーバーの開始および停止、アプリケーションのデプロイおよびアンデプロイ、システム設定の調整、サーバー設定への永続的な変更を行うために使用できます。管理コンソールは、管理タスクを実行する機能も持ち、変更の反映にサーバーインスタンスの再起動またはリロードが必要な場合はライブ通知も行います。管理対象ドメインでは、同じドメインのサーバーインスタンスとサーバーグループをドメインコントローラーの管理コンソールから集中管理できます。

管理 CLI は、JBoss EAP のコマンドラインの管理ツールです。管理 CLI は、サーバーの開始および停止、アプリケーションのデプロイおよびアンデプロイ、システム設定の指定、およびその他の管理タスクの実行に使用できます。管理 CLI は、管理対象ドメインのドメインコントローラーに接続し、ドメインで管理操作を実行することもできます。管理 CLI は、web ベースの管理ツールが実行できるタスクをすべて実行でき、web ベースの管理ツールでは利用できない多くの細かな低レベル操作も実行できます。



注記

JBoss EAP に含まれる API を使用すると、JBoss EAP に同梱されるクライアントだけでなく、他のクライアントを作成して HTTP またはネイティブインターフェース上で管理インターフェースを呼び出しすることができます。

2.3. JMX

JMX (Java Management Extensions) は、JDK およびアプリケーション管理操作をリモートで行う方法を提供します。JBoss EAP の管理 API は、JMX 管理 Bean として公開されます。これらの管理 Bean はコア MBean と呼ばれ、この Bean へのアクセスは基盤の管理 API と全く同じように制御およびフィルターされます。

管理 CLI と管理コンソールの他に、JMX 公開 Bean は管理操作にアクセスし、実行するための代替メカニズムです。

2.4. ロールベースのアクセス制御

ロールベースのアクセス制御 (RBAC) は管理ユーザーにパーミッションを指定するメカニズムです。JBoss EAP サーバーの管理責任を複数のユーザーに分担でき、各ユーザーは無制限のアクセスを必要としません。JBoss EAP では、管理ユーザーの役割を分離することで、不必要な権限を与えずに、組織の個人やグループ間で責任を簡単に分散できます。これにより、設定、デプロイメント、および管理の柔軟性を維持しながらサーバーやデータのセキュリティーを可能な限り最大限にします。

JBoss EAP の RBAC は、ロールパーミッションと制約の組み合わせによって動作します。事前定義されたロールが 7 つ用意され、各ロールは異なる固定のパーミッションを持ちます。各管理ユーザーには、サーバーの管理時に許可される動作を指定する 1 つ以上のロールが割り当てられます。

JBoss EAP では、RBAC はデフォルトで無効になっています。

標準のロール

JBoss EAP には、**Monitor**、**Operator**、**Maintainer**、**Deployer**、**Auditor**、**Administrator**、および **SuperUser** の 7 つユーザーロールが事前定義されています。各ロールは、異なるパーミッションのセットを持ち、特定のユースケースに対応します。パーミッションの数は、**Monitor**、**Operator**、**Maintainer**、**Administrator**、**SuperUser** ロールの順に多くなり、それぞれその前のロールのパーミッションを持ちます。**Auditor** ロールは **Monitor** ロール、**Deployer** ロールは **Maintainer** ロールと似ていますが、特別なパーミッションを持ち、制限が適用されます。

Monitor

Monitor ロールは最も少ないパーミッションを持ち、現在の設定とサーバーの状態のみを読み取りできます。このロールは、サーバーのパフォーマンスを追跡および報告する必要があるユーザーを対象としています。**Monitor** ロールを持つユーザーはサーバー設定を変更できず、機密のデータや操作にもアクセスできません。

Operator

Operator ロールは **Monitor** ロールのパーミッションに加え、サーバーのランタイム状態を変更することができます。このため、**Operator** ロールを持つユーザーはサーバーのリロードおよびシャットダウンを実行でき、JMS 宛先の一時停止および再開も実行できます。**Operator** ロールは、必要時にサーバーを確実にシャットダウンおよび再起動できるため、アプリケーションサーバーの物理または仮想ホストの責任者に適したロールです。**Operator** ロールを持つユーザーはサーバー設定を変更できず、機密のデータや操作にもアクセスできません。

Maintainer

Maintainer ロールは、ランタイム状態と、機密データおよび機密操作を除くすべての設定を表示および変更できます。**Maintainer** ロールは機密データと機密操作にアクセスできない汎用のロールです。**Maintainer** ロールを持つユーザーには、パスワードやその他の機密情報へのアクセスを許可せずに、サーバー管理に必要なほぼ完全なアクセス権利を付与することができます。**Maintainer** は機密のデータや操作にはアクセスできません。

Administrator

Administrator ロールは、監査ロギングシステムを除くサーバーのすべてのリソースおよび操作に無制限にアクセスできます。**Administrator** ロールは機密のデータや操作にアクセスできます。また、このロールはアクセス制御システムも設定できます。**Administrator** ロールは、機密データを扱う場合やユーザーやロールを設定する場合のみ必要となります。**Administrator** は監査ロギングシステムにはアクセスできず、**Auditor** または **SuperUser** ロールに変わることはできません。

SuperUser

SuperUser ロールには制限がなく、監査ロギングシステムと機密データを含むサーバーのすべてのリソースと操作に完全アクセスできます。RBAC が無効の場合、すべての管理ユーザーは **SuperUser** ロールと同等のパーミッションを持ちます。

Deployer

Deployer ロールは **Monitor** ロールと同じパーミッションを持ちますが、デプロイメントと、アプリケーションリソースとして有効になっている他のリソースタイプの設定および状態を変更できます。

Auditor

Auditor ロールは **Monitor** ロールのパーミッションをすべて持ちます。機密データも閲覧できますが、変更はできません。監査ロギングシステムに完全アクセスできます。**SuperUser** 以外のロールで監査ロギングシステムにアクセスできるのは **Auditor** ロールのみです。 **Auditor** は機密データやリソースを変更できず、読み取りのみが許可されます。

Permissions

Permissions は各ロールの権限を決定します。すべてのロールにすべてのパーミッションがあるわけではありません。**SuperUser** にはすべてのパーミッションがあり、**Monitor** のパーミッションは最も少なくなります。各パーミッションは、リソースの単一のカテゴリーへの読み取りアクセスや書き込みアクセスを付与できます。カテゴリーはランタイム状態、サーバー設定、機密データ、監査ログ、およびアクセス制御システムになります。

表2.1 各 Monitor、Operator、Maintainer、および Deployer ロールのパーミッション

	Monitor	Operator	Maintainer	Deployer
設定と状態の読み取り	✘	✘	✘	✘
機密データの読み取り [2]				
機密データの変更 [2]				
監査ログの読み取り/変更				
ランタイム状態の変更		✘	✘	✘[1]
永続化設定の変更			✘	✘[1]

	Monitor	Operator	Maintainer	Deployer
アクセス制御の読み取り/変更				

[1] パーミッションはアプリケーションリソースに制限されます。

表2.2 各 Auditor、Administration、および SuperUser ロールのパーミッション

	Auditor	Administrator	SuperUser
設定と状態の読み取り	✘	✘	✘
機密データの読み取り [2]	✘	✘	✘
機密データの変更 [2]		✘	✘
監査ログの読み取り/変更	✘		✘
ランタイム状態の変更		✘	✘
永続化設定の変更		✘	✘
アクセス制御の読み取り/変更		✘	✘

[2] 機密データとして考慮されるリソースは機密性を使用して設定されます。

制約

制約とは、指定のリソースリストに対するアクセス制御設定の名前付きセットです。RBAC システムは制約とロールパーミッションの組み合わせを使用して、特定ユーザーが管理操作を実行できるかどうかを決定します。

制約は 3 つの種類に分類されます。

アプリケーション制約

アプリケーション制約は、Deployer ユーザーがアクセスできるリソースおよび属性を定義します。デフォルトで有効になっているアプリケーション制約はコアのみで、デプロイメントとデプロイメントオーバーレイが含まれます。アプリケーション制約には、データソース、ロギング、メール、メッセージング、ネーミング、リソースアダプター、およびセキュリティーも含まれますが、デフォルトでは有効になっていません。これらの制約によって、Deployer ユーザーはアプリケーションをデプロイできるだけでなく、これらのアプリケーションが必要とするリソースを設定および維持できます。

機密性制約

機密性制約は、機密として考慮されるリソースを定義します。通常、機密リソースとは、パスワードなどの秘密のリソースや、ネットワーキング、JVM 設定、システムプロパティーなどのサーバーの操作に重大な影響を与えるリソースのことです。アクセス制御システム自体も機密であると見な

されます。機密リソースへの書き込みが許可されるロールは Administrator と SuperUser ロールのみです。Auditor ロールは機密リソースの読み取りのみが許可されます。その他のロールは機密リソースにアクセスできません。

vault 式制約

vault 式制約は、vault 式の読み書きが機密操作として考慮されるかどうかを定義します。デフォルトでは、vault 式の読み書きは機密操作です。

2.5. 宣言型セキュリティーと JAAS

宣言型セキュリティーは、コンテナーを使用してセキュリティーを管理することでアプリケーションコードからセキュリティーの懸念を分離する方法です。コンテナーはファイルパーミッション、またはユーザー、グループ、およびロールを基に承認システムを提供します。通常この方法は、アプリケーション自体にセキュリティーの責任をすべて与えるプログラムによるセキュリティーよりも優れています。JBoss EAP は **security** サブシステムでセキュリティードメインを使用して宣言型セキュリティーを提供します。

JAAS (Java Authentication and Authorization Service) は、ユーザー認証および承認用の Java パッケージで構成される宣言型セキュリティー API です。この API は 標準の PAM (Pluggable Authentication Modules) フレームワークの Java 実装です。Java EE アクセス制御アーキテクチャーを拡張し、ユーザーベースの承認をサポートします。 JBoss EAP の **security** サブシステムは実際に JAAS API をベースにしています。

JAAS は **security** サブシステムの基盤であるため、認証はプラグ可能な方法で実行されます。これにより、Java アプリケーションは Kerberos や LDAP などの基礎の認証技術から独立でき、セキュリティーマネージャーは異なるセキュリティーインフラストラクチャーで機能できます。セキュリティーマネージャーの実装を変更しなくてもセキュリティーインフラストラクチャーとの統合を実現できます。JAAS が使用する認証スタックの設定のみを変更する必要があります。

2.6. ELYTRON サブシステム

elytron サブシステムは JBoss EAP 7.1 で新規導入されました。これは、アプリケーションサーバー全体でのセキュリティーの統一に使用されるセキュリティーフレームワークである WildFly Elytron プロジェクトをベースにしています。**elytron** サブシステムにより、単一の設定場所でアプリケーションと管理インタフェースの両方をセキュアにできます。WildFly Elytron は、機能のカスタム実装を提供し、**elytron** サブシステムと統合するために複数の API と SPI も提供します。

他にも、WildFly Elytron には複数の重要な機能があります。

- 強化された HTTP および SASL 認証の認証メカニズム。
- セキュリティードメイン全体で **SecurityIdentities** が伝搬されるようにする改良されたアーキテクチャー。これにより、透過的な変換を承認に使用できる状態にします。変換は、設定可能なロールデコーダー、ロールマッパー、およびパーミッションマッパーを使用して実行されます。
- 暗号化スイートおよびプロトコルを含む SSL/TTS 設定の一元化。
- 一括 **SecureIdentity** 構築などの SSL/TLS の最適化と、確立中の SSL/TLS 接続への承認の密な結び付け。一括 **SecureIdentity** 構築によって、リクエストごとに **SecureIdentity** を構築する必要がなくなります。確立中の SSL/TLS 接続に認証を密に結び付けると、最初のリクエストを受け取る **前** にパーミッションチェックが行われるようになります。
- プレーンテキストの文字列を格納する、以前の vault 実装に取って代わるセキュアクレデンシャルストア。

新しい **elytron** サブシステムは、レガシーの **security** サブシステムとレガシーのコア管理認証と並行して存在します。レガシーと Elytron の両方は、管理インターフェースのセキュア化に使用され、アプリケーションにセキュリティーを提供するためにも使用されます。

重要

The architectures of Elytron and the legacy security subsystem that is based on PicketBox are very different. With Elytron, an attempt was made to create a solution that allows you to operate in the same security environments in which you currently operate; however, this does **not** mean that every PicketBox configuration option has an equivalent configuration option in Elytron.

If you are not able to find information in the documentation to help you achieve similar functionality using Elytron that you had when using the legacy security implementation, you can find help in one of the following ways.

- If you have a Red Hat Development subscription, you have access to Support Cases, Solutions, and Knowledge Articles on the Red Hat Customer Portal. You can also open a case with Technical Support and get help from the WildFly community as described below.
- If you do not have a Red Hat Development subscription, you can still access Knowledge Articles on the Red Hat Customer Portal. You can also join the user forums and live chat to ask questions of the WildFly community. The WildFly community offerings are actively monitored by the Elytron engineering team.

2.6.1. 中核の概念およびコンポーネント

小さいコンポーネントを使用して完全なセキュリティーポリシーを組み立てるのが、**elytron** サブシステムのアーキテクチャーやデザインの背後にある概念です。デフォルトでは、JBoss EAP は多くのコンポーネントの実装を提供しますが、**elytron** サブシステムは特殊なカスタム実装の提供も可能にします。

elytron サブシステムのコンポーネントの各実装は、個別の性能として処理されます。そのため、異なるリソースを使用して異なる実装を組み合わせ、モデル化できます。

2.6.1.1. 性能および要件

性能 (capability) とは、JBoss EAP で使用される機能の一部で、管理レイヤーを使用して公開されます。性能は他の性能に依存することもでき、この依存関係は管理レイヤーによって仲介されます。一部の性能は JBoss EAP によって自動的に提供されますが、起動時に利用可能な性能の完全セットは、JBoss EAP の設定を使用して判断されます。管理レイヤーは、サーバーの起動中と設定の変更時に、性能に必要な別の性能がすべて存在することを検証します。性能は JBoss Modules および拡張と統合しますが、これらの概念はすべて異なります。

性能は、依存する他の性能を登録する他に、依存する性能に関連する要件を登録する必要もあります。性能は以下のような要件を指定できます。

要件

性能が機能するには他の性能に依存する必要があるため、依存する性能が常に存在する必要があります。

オプションの要件

性能のオプションは、有効化が可能または不可能な別の性能に依存します。そのため、設定が分析されるまで要件を判断できないか、要件が分かりません。

起動時のみの要件

性能は他に必要な性能が実行時に存在するかどうかをチェックします。必要な性能が存在する場合は使用され、必要な性能が存在しない場合は使用されません。

性能と要件の詳細は、WildFly ドキュメント を参照してください。

2.6.1.2. API、SPI、およびカスタム実装

Elytron はセキュリティー API および SPI を提供します。他のサブシステムやコンシューマーはそれらの API や SPI を直接使用できるため、統合のオーバーヘッドが削減されます。ほとんどのユーザーは JBoss EAP で提供される機能を使用しますが、Elytron API および SPI は Elytron の機能を置き換えまたは拡張するためにカスタム実装によっても使用されます。

2.6.1.3. セキュリティードメイン

セキュリティードメインは、1 つ以上のセキュリティーレルムや変換を実行するリソースのセットが関係するセキュリティーポリシーを表します。セキュリティードメインは **SecurityIdentity** を作成します。**SecurityIdentity** は、アプリケーションなど、承認を実行する他のリソースによって使用されます。**SecurityIdentity** は、未処理の **AuthorizationIdentity** とその関連するロールやパーミッションを基にした現在のユーザーを表します。

セキュリティードメインを設定して、別のセキュリティードメインから **SecurityIdentity** の **インフロー**を許可することもできます。アイデンティティーが**インフロー**すると、元の処理されていない **AuthorizationIdentity** を保持し、新しいロールとパーミッションのセットが割り当てられ、新しい **SecurityIdentity** が作成されます。

重要

Elytron セキュリティードメインの使用は、ドメインごとに 1 つに限定されます。複数のレガシーセキュリティーが必要である場合、1 つの Elytron セキュリティードメインを使用して対応できるようになりました。

2.6.1.4. セキュリティーレルム

セキュリティーレルムはアイデンティティーストアへのアクセスを提供し、クレデンシャルの取得に使用されます。これらのクレデンシャルにより、認証メカニズムは認証を実行するために未処理の **AuthorizationIdentity** を取得できます。また、これらのクレデンシャルは、証拠の検証時に認証メカニズムが検証を行えるようにします。

1 つ以上のセキュリティーレルムを 1 つのセキュリティードメインに関連付けることができます。また、一部のセキュリティー実装は変更のために API も公開するため、セキュリティーレルムは基盤のアイデンティティーストアへの更新を作成できます。

2.6.1.5. ロールデコーダー

ロールデコーダーはセキュリティードメインに関連付けられ、現ユーザーのロールをデコードするために使用されます。ロールデコーダーは、セキュリティーレルムから返される未処理の **AuthorizationIdentity** を取り、その属性をロールに変換します。

2.6.1.6. ロールマッパー

ロールマッパーはロールの変更をアイデンティティーに適用します。これは、ロールの形式の正規化から特定ロールの追加や削除までさまざまです。ロールマッパーはセキュリティーレルムとセキュリティードメインの両方に関連付けることができます。ロールマッパーがセキュリティーレルムに関連

付けられた場合、セキュリティードメインレベルでロールのデコードや追加のロールマッピングなどの変換が発生する前に、マッピングがセキュリティーレルムレベルで適用されます。ロールマッパーと他の変換 (ロールデコーダーなど) が両方セキュリティードメインで設定された場合、ロールマッパーが適用される前に他の変換がすべて実行されます。

2.6.1.7. パーミッションマッパー

パーミッションマッパーはセキュリティードメインと関連付けられ、パーミッションを **SecurityIdentity** に割り当てます。

2.6.1.8. プリンシパルトランスフォーマー

プリンシパルトランスフォーマーは、 **elytron** サブシステム内の複数の場所で使用できます。プリンシパルトランスフォーマーは名前を別の名前に変換またはマップできます。

2.6.1.9. プリンシパルデコーダー

プリンシパルデコーダーは、 **elytron** サブシステム内の複数の場所で使用できます。プリンシパルデコーダーはアイデンティティーを **Principal** から名前の文字列表現に変換します。たとえば、**X500PrincipalDecoder** を使用すると **X500Principal** を証明書の識別名から文字列表現に変換できます。

2.6.1.10. レルムマッパー

レルムマッパーはセキュリティードメインと関連付けられ、セキュリティードメインに複数のセキュリティーレルムが設定されている場合に使用されます。レルムマッパーは、**http-authentication-factory** および **sasl-authentication-factory** の **mechanism** または **mechanism-realm** に関連付けられることもあります。レルムマッパーは、認証中に提供された名前を使用して認証のセキュリティーレルムを選択し、未処理の **AuthorizationIdentity** を取得します。

2.6.1.11. 認証ファクトリー

認証ファクトリーは、認証ポリシーを表します。認証はセキュリティードメイン、メカニズムファクトリー、およびメカニズムセレクターと関連付けられます。セキュリティードメインは、認証される **SecurityIdentity** を提供します。メカニズムファクトリーはサーバー側の認証メカニズムを提供します。メカニズムセレクターは、選択したメカニズムに固有する設定を取得します。メカニズムセレクターには、メカニズムがリモートクライアントに提示する必要のあるレルム名に関する情報や、認証プロセス中に使用する追加のプリンシパルトランスフォーマーとレルムマッパーに関する情報を含めることができます。

2.6.1.12. キーストア

key-store は、キーストアの種類、その場所、アクセスするためのクレデンシャルなどを含むキーストアまたはトラストストアの定義です。

2.6.1.13. キーマネージャー

key-manager は **key-store** を参照し、SSL コンテキストとともに使用されます。

2.6.1.14. トラストマネージャー

trust-manager は、**key-store** に定義されるトラストストアを参照します。通常は、双方向 SSL/TLS の SSL コンテキストとともに使用されます。

2.6.1.15. SSL コンテキスト

elytron サブシステム内で定義される SSL コンテキストは**javax.net.ssl.SSLContext** で、SSL コンテキストを直接使うものが使用できます。 SSL コンテキストの通常の設定の他に、暗号化スイートやプロトコルなどの追加項目を設定することが可能です。SSL コンテキストは、設定された追加項目をラップします。

2.6.1.16. セキュアなクレデンシャルストア

プレーンテキストの文字列の暗号化に使用されたこれまでの vault 実装は、新設計のクレデンシャルストアに置き換えられました。クレデンシャルストアは、保存するクレデンシャルの保護の他に、プレーンテキストの文字列を保存するために使用されます。

2.6.2. Elytron 認証プロセス

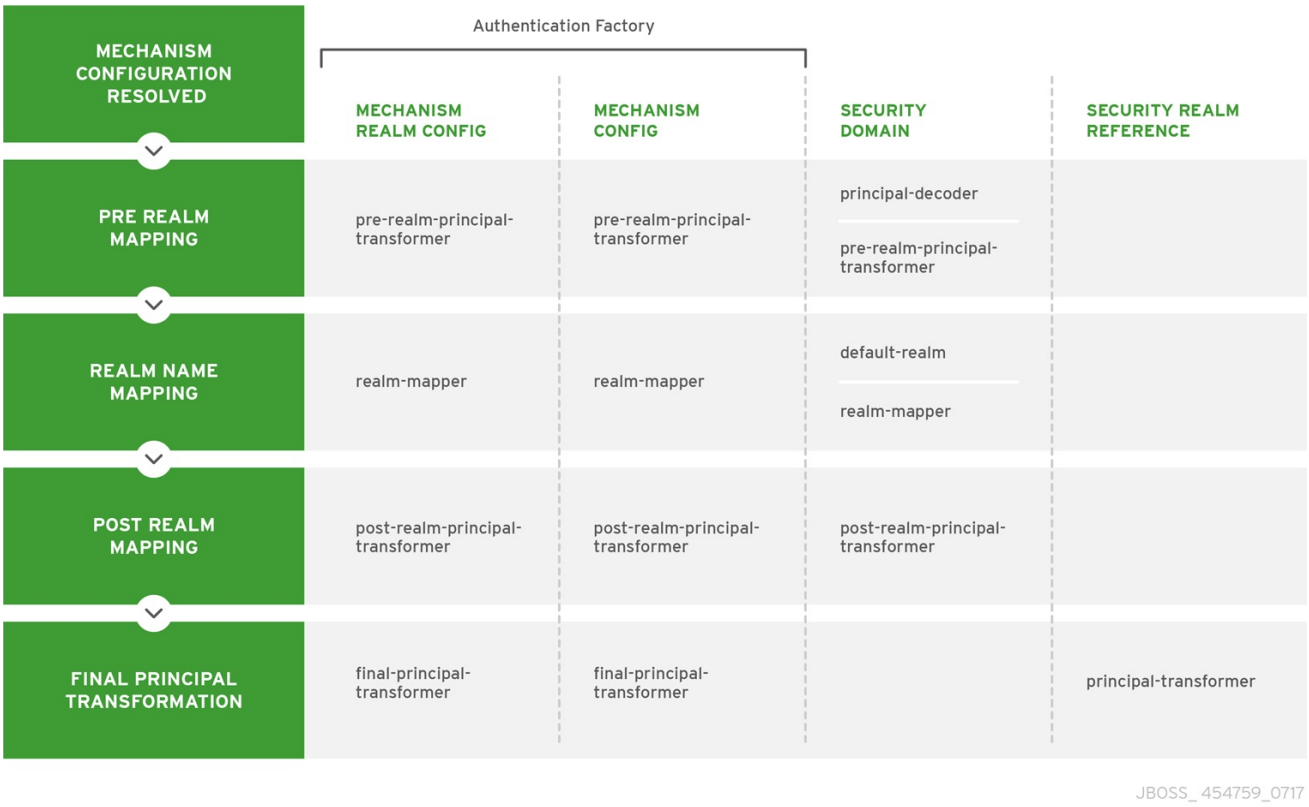
複数のプリンシパルトランスフォーマー、レルムマッパー、およびプリンシパルデコーダーは、**elytron** サブシステム内に定義できます。以下のセクションでは、認証プロセス中にこれらのコンポーネントどのように機能するかを説明し、さらにプリンシパルが適切なセキュリティーレルムにマップされる方法についても説明します。

プリンシパルが認証されると、以下の手順が順番に実行されます。

1. 適切なメカニズムの構成が判断および設定されます。
2. 受信プリンシパルが **SecurityIdentity** へマップされます。
3. 適切なセキュリティーレルムを判断するため、この **SecurityIdentity** が使用されます。
4. セキュリティーレルムが特定された後にプリンシパルが再度変換されます。
5. メカニズム固有の変換を可能にするため、最終の変換が 1 度発生します。

以下の図は上記の手順を表しています。左側の緑の部分は各手順を示し、右側は各段階で使用されるコンポーネントを表しています。

図2.1 Elytron 認証プロセス



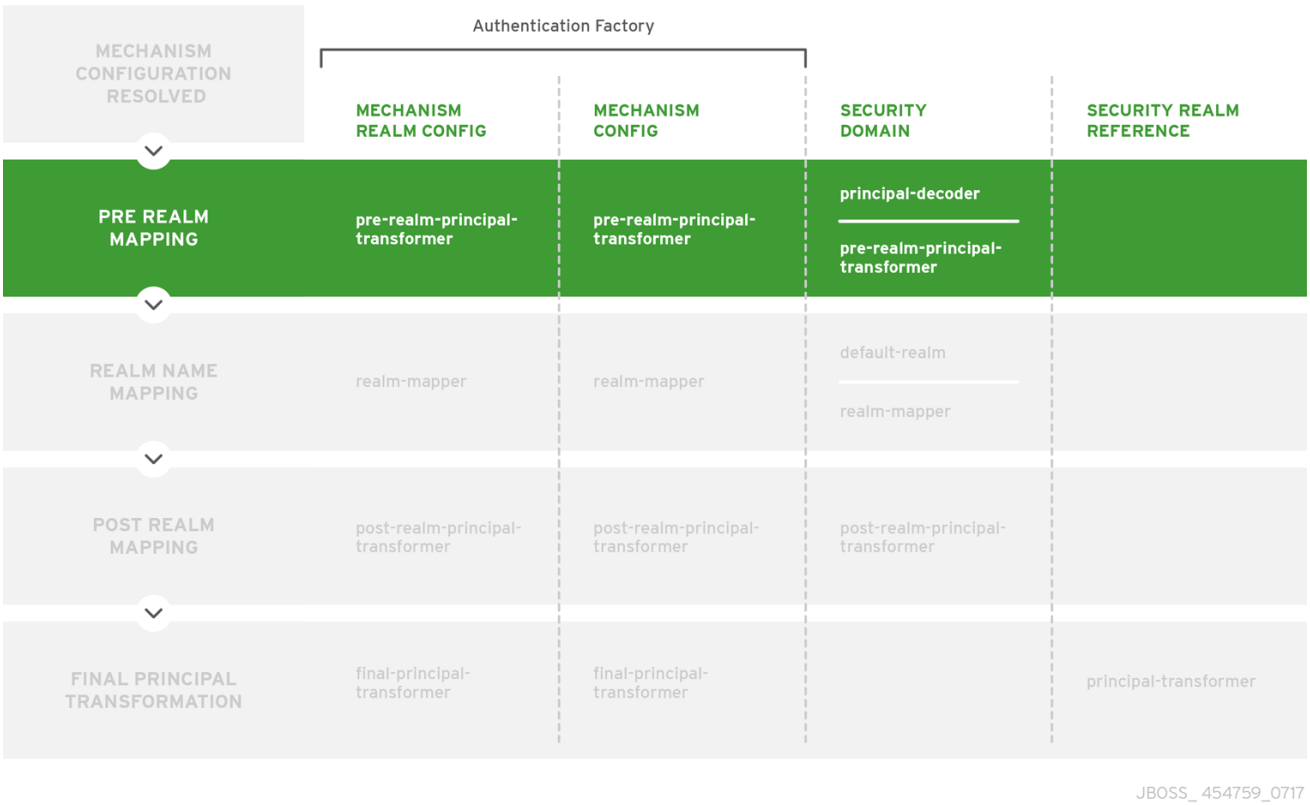
レルムの前のマッピング

レルムの前のマッピングが行われる間、認証されたプリンシパルは **SecurityIdentity** にマップされます。**SecurityIdentity** は使用されるセキュリティーレルムを特定できるフォームで、認証された情報を表す単一の **Principal** を含みます。プリンシパルトランスフォーマーとプリンシパルデコーダーは次の順番で呼び出されます。

1. メカニズムレルム - **pre-realm-principal-transformer**
2. メカニズム設定 - **pre-realm-principal-transformer**
3. セキュリティードメイン - **principal-decoder** および **pre-realm-principal-transformer**

この手順によって null プリンシパルが発生した場合、エラーが発生し、認証は強制終了されます。

図2.2 レルムの前のマッピング

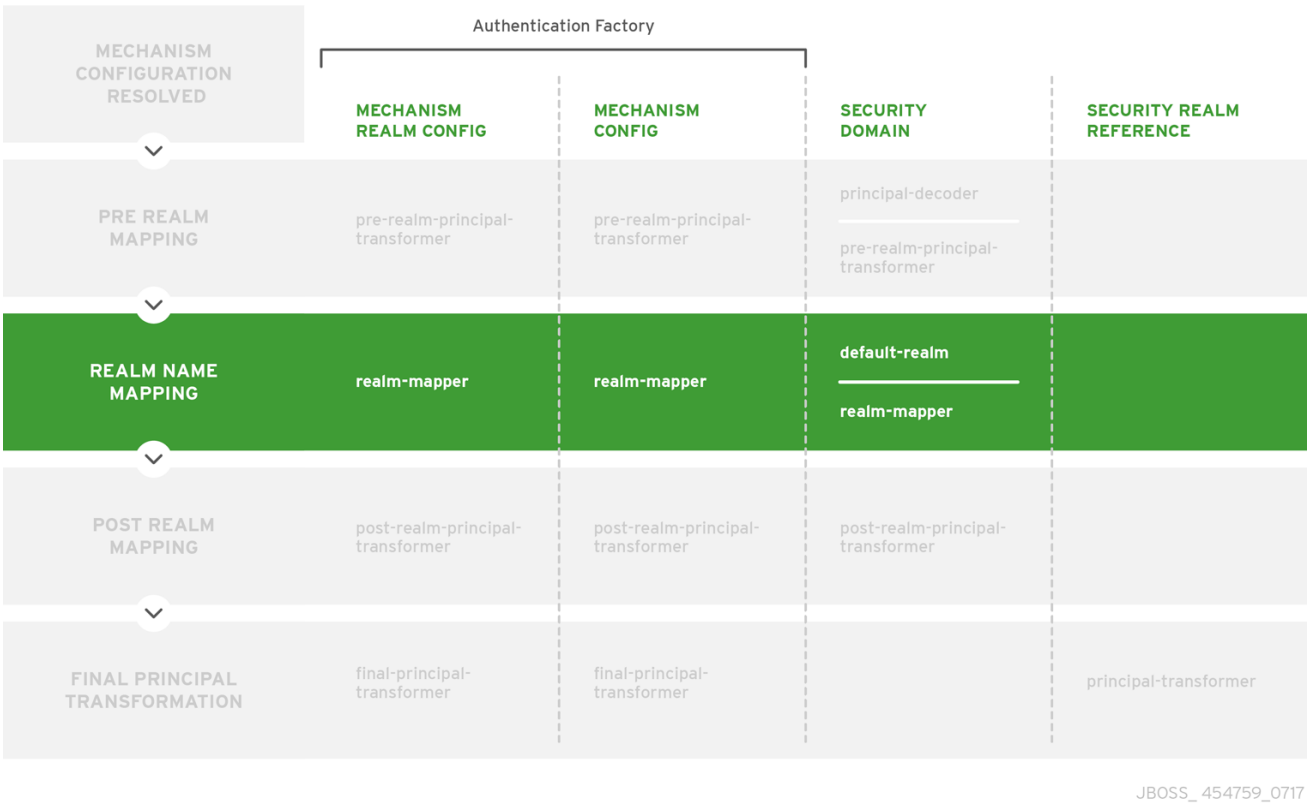


レルム名マッピング
マップされたプリンシパルが取得されると、アイデンティティーのロードに使用されるセキュリティーレルムが特定されます。この時点では、レルム名はセキュリティードメインによって参照され、セキュリティーレルムによって定義される名前で、メカニズムレルム名ではありません。設定は、セキュリティーレルム名を次の順序で検索します。

1. メカニズムレルム - **`realm-mapper`**
2. メカニズム設定- **`realm-mapper`**
3. セキュリティードメイン - **`realm-mapper`**

`RealmMapper` が null を返す場合や、利用できるマッパーがない場合、セキュリティードメインの **`default-realm`** が使用されます。

図2.3 レルム名マッピング

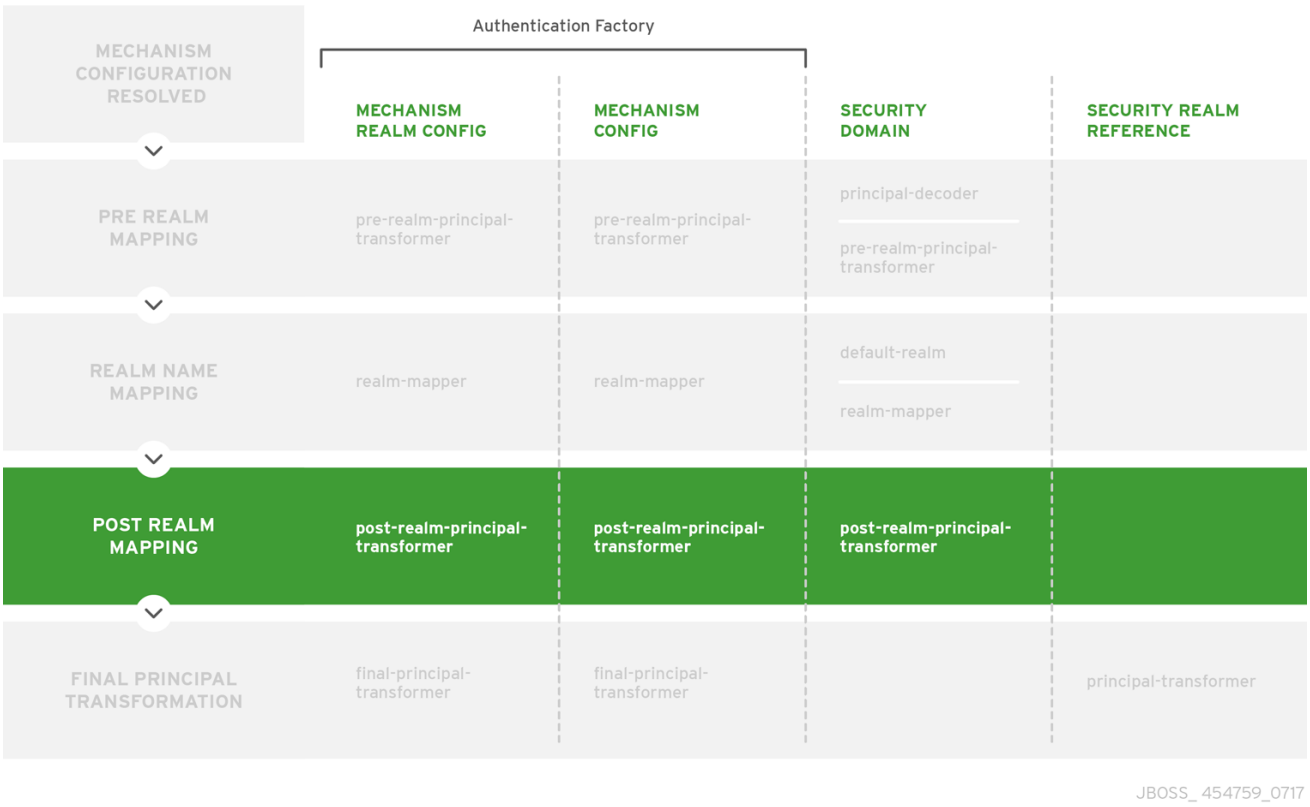


レルムの後のマッピング
レルムの特定後、プリンシパルに対して次の変換が実行されます。変換は次の順番で呼び出しされます。

1. メカニズムレルム - **`post-realm-principal-transformer`**
2. メカニズム設定 - **`post-realm-principal-transformer`**
3. セキュリティードメイン - **`post-realm-principal-transformer`**

この手順によって null プリンシパルが発生した場合、エラーが発生し、認証は強制終了されます。

図2.4 レルムの後のマッピング

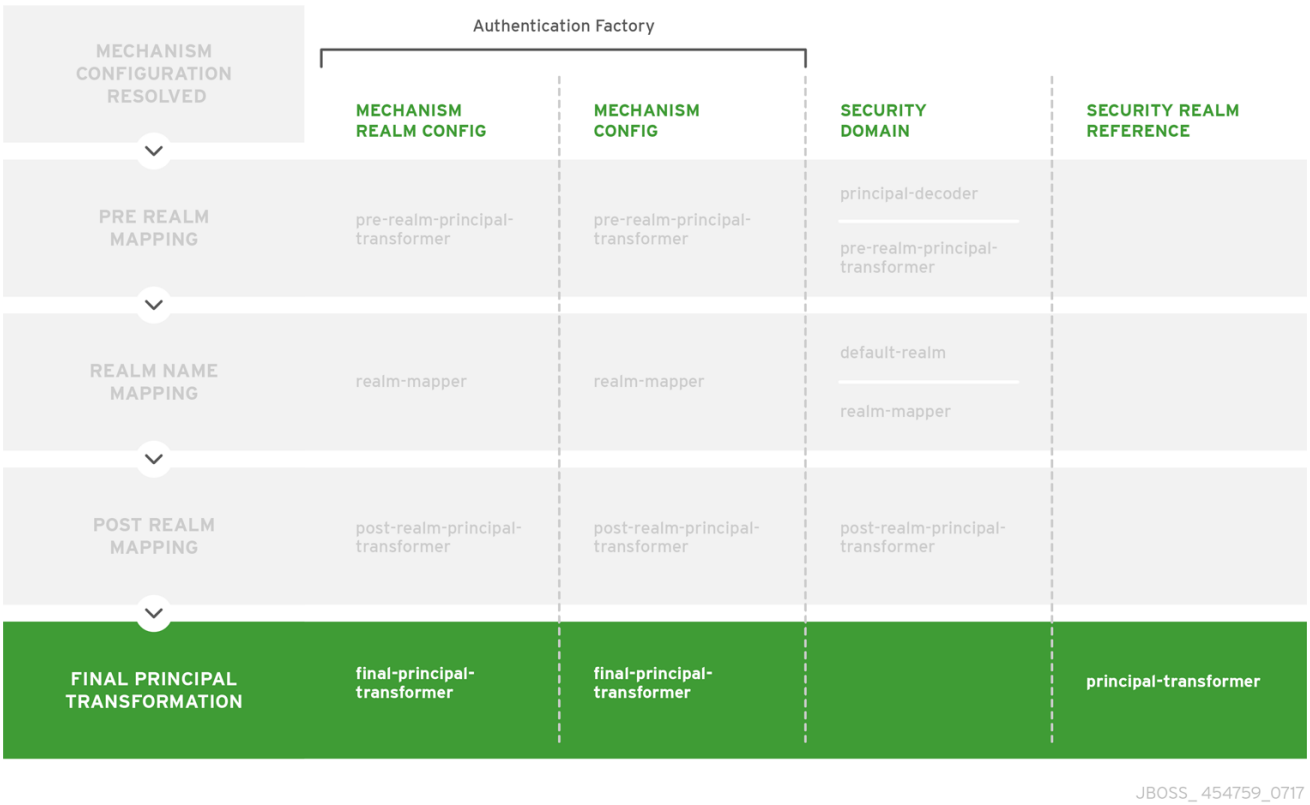


最終のプリンシパル変換
最後に、メカニズム固有の変換がドメイン固有の変換の前および後に適用されるようにするため、最後のプリンシパル変換が発生します。これが必要でない場合、レルムの後のマッピングの段階で同じ結果が得られます。変換は次の順序で呼び出しされます。

1. メカニズムレルム - **`final-principal-transformer`**
2. メカニズム設定 - **`final-principal-transformer`**
3. レルムマッピング - **`principal-transformer`**

この手順によって null プリンシパルが発生した場合、エラーが発生し、認証は強制終了されます。

図2.5 最終のプリンシパル変換



レルムアイデンティティーの取得
最終のプリンシパル変換の後、認証の継続に使用されるレルムアイデンティティーを取得するため、レルム名マッピング で特定されたセキュリティーレルムが呼び出しされます。

2.6.3. HTTP 認証

Elytron は、**BASIC**、**FORM**、**DIGEST**、**SPNEGO**、および **CLIENT_CERT** を含む完全な HTTP 認証メカニズムを提供します。HTTP 認証は、**HttpAuthenticationFactory** を使用して処理されます。**HttpAuthenticationFactory** は HTTP 認証メカニズムを使用するための認証ポリシーであり、設定された認証メカニズムのファクトリーでもあります。

HttpAuthenticationFactory は以下を参照します。

SecurityDomain
メカニズム認証が実行されるセキュリティードメイン。

HttpServerAuthenticationMechanismFactory
サーバー側 HTTP 認証メカニズムの一般的なファクトリー。

MechanismConfigurationSelector
これを使用して認証メカニズムの追加設定を提供できます。**MechanismConfigurationSelector** の目的は、選択したメカニズムに固有の設定を取得することです。これには、メカニズムがリモートクライアントに提示する必要のあるレルム名、追加のプリンシパルトランスフォーマー、認証プロセス中に使用するレルムマッパーなどに関する情報が含まれます。

2.6.4. SASL 認証

SASL は、認証メカニズム自体を使用するプロトコルから分離するフレームワークです。また、**DIGEST-MD5**、**GSSAPI**、**OTP**、**SCRAM** などの追加の認証メカニズムも使用できます。SASL 認証

は Java EE 7 の一部ではありません。SASL 認証は、**SaslAuthenticationFactory** を使用して処理され、**SaslAuthenticationFactory** は SASL 認証メカニズムを使用するための認証ポリシーであり、設定された認証メカニズムのファクトリーでもあります。

SaslAuthenticationFactory は以下を参照します。

SecurityDomain

メカニズム認証が実行されるセキュリティードメイン。

SaslServerFactory

サーバー側 SASL 認証メカニズムの一般的なファクトリー。

MechanismConfigurationSelector

これを使用して認証メカニズムの追加設定を提供できます。**MechanismConfigurationSelector** の目的は、選択したメカニズムに固有の設定を取得することです。これには、メカニズムがリモートクライアントに提示する必要のあるレルム名、追加のプリンシパルトランスフォーマー、認証プロセス中に使用するレルムマッパーなどに関する情報が含まれます。

2.6.5. Elytron サブシステムとレガシーシステム間の対話

レガシー **security** サブシステムコンポーネントおよびレガシーコア管理認証両方の主なコンポーネントの一部を Elytron の性能にマップできます。これにより、これらのレガシーコンポーネントを Elytron ベースの設定で使用でき、レガシーコンポーネントから増分移行を行うことができます。

2.6.6. Elytron サブシステムのリソース

JBoss EAP は、**elytron** サブシステムで以下のリソースを提供します。

- ファクトリー
- プリンシパルトランスフォーマー
- プリンシパルデコーダー
- レルムマッパー
- レルム
- パーミッションマッパー
- ロールデコーダー
- ロールマッパー
- SSL コンポーネント
- その他

ファクトリー

aggregate-http-server-mechanism-factory

HTTP サーバーファクトリーが他の HTTP サーバーファクトリーの集約である、HTTP サーバーファクトリー定義。

aggregate-sasl-server-factory

SASL サーバーファクトリーが他の SASL サーバーファクトリーの集約である、SASL サーバーファクトリー定義。

configurable-http-server-mechanism-factory

別の HTTP サーバーファクトリーをラッピングし、指定の設定とフィルタリングを適用する HTTP サーバーファクトリー定義。

configurable-sasl-server-factory

別の SASL サーバーファクトリーをラッピングし、指定の設定とフィルタリングを適用する SASL サーバーファクトリー定義。

custom-credential-security-factory

カスタムクレデンシャルの **SecurityFactory** 定義。

http-authentication-factory

セキュリティードメインと **HttpServerAuthenticationMechanismFactory** の関連が含まれるリソース。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with Certificates」を参照してください。

kerberos-security-factory

認証中に使用する **GSSCredential** を取得するためのセキュリティーファクトリー。
詳細は、JBoss EAP『**How to Set Up SSO with Kerberos** 』の「Configure the Elytron Subsystem」を参照してください。

mechanism-provider-filtering-sasl-server-factory

プロバイダーを使用してファクトリーがロードされた場合にプロバイダーによるフィルタリングを有効にする SASL サーバーファクトリー定義。

provider-http-server-mechanism-factory

HTTP サーバーファクトリーがプロバイダーリストからのファクトリーの集約である、HTTP サーバーファクトリー定義

provider-sasl-server-factory

SASL サーバーファクトリーがプロバイダーリストからのファクトリーの集約である、SASL サーバーファクトリー定義。

sasl-authentication-factory

セキュリティードメインと SASL サーバーファクトリーの関連が含まれるリソース。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Secure the Management Interfaces with a New Identity Store」を参照してください。

service-loader-http-server-mechanism-factory

HTTP サーバーファクトリーが **ServiceLoader** を使用して特定されるファクトリーの集約である、HTTP サーバーファクトリー定義。

service-loader-sasl-server-factory

SASLサーバーファクトリーが **ServiceLoader** を使用して特定されるファクトリーの集約である、SASL サーバーファクトリー定義。

プリンシパルトランスフォーマー

aggregate-principal-transformer

個別のトランスフォーマーは、トランスフォーマーの 1 つが null でないプリンシパルを返すまで元のプリンシパルを変換しようとします。

chained-principal-transformer

プリンシパルトランスフォーマーが他のプリンシパルトランスフォーマーをチェーンするプリンシパルトランスフォーマー定義。

constant-principal-transformer

プリンシパルトランスフォーマーが常に同じ定数を返す、プリンシパルトランスフォーマー定義。

custom-principal-transformer

カスタムプリンシパルトランスフォーマー定義。

regex-principal-transformer

正規表現ベースのプリンシパルトランスフォーマー。

regex-validating-principal-transformer

正規表現を使用して名前を検証する、正規表現をベースとしたプリンシパルトランスフォーマー。

プリンシパルデコーダー

aggregate-principal-decoder

プリンシパルデコーダーが他のプリンシパルデコーダーの集約であるプリンシパルデコーダー定義。

concatenating-principal-decoder

プリンシパルデコーダーが他のプリンシパルデコーダーの連結であるプリンシパルデコーダー定義。

constant-principal-decoder

常に同じ定数を返すプリンシパルデコーダーの定義。

custom-principal-decoder

カスタムプリンシパルデコーダーの定義。

x500-attribute-principal-decoder

X500 属性ベースのプリンシパルデコーダーの定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with Certificates」を参照してください。

レルムマッパー

constant-realm-mapper

常に同じ値を返す定数のレルムマッパー の定義。

custom-realm-mapper

カスタムレルムマッパーの定義。

mapped-regex-realm-mapper

最初に正規表現を使用してレルム名を抽出した後にレルム名の設定済みマッピングを使用して変換されるレルムマッパー実装の定義。

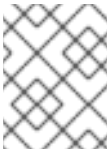
simple-regex-realm-mapper

正規表現からのキャプチャーグループを使用してレルム名の抽出を試みる簡単なレルムマッパーの定義。一致するものを提供しない場合、代わりに委譲マッパーが使用されます。

レルム

aggregate-realm

2 つのレルム (認証ステップのレルムおよび承認ステップのアイデンティティーをロードするレルム) の集約であるレルム定義。



注記

aggregate-realm の承認ステップで、エクスポートされたレガシーセキュリティードメインを Elytron セキュリティーレルムとして使用できません。

caching-realm

他のセキュリティーレルムへのキャッシュを可能にするレルム定義。キャッシュストラテジーは **LRU** で、エントリーの最大数に達したときにアクセスが最も少ないエントリーが破棄されます。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Set Up Caching for Security Realms」を参照してください。

custom-modifiable-realm

変更可能として設定されたカスタムレルムは、**ModifiableSecurityRealm** インターフェースを実装することが想定されます。レルムを変更可能として設定すると、管理操作を利用してレルムを操作できます。

custom-realm

カスタムレルム定義は **SecurityRealm** インターフェースまたは **ModifiableSecurityRealm** インターフェースを実装できます。どちらのインターフェースが実装されたかに関わらず、レルムの管理に管理操作は公開されません。しかし、レルムに依存するその他のサービスは型チェックやキャストを実行して変更 API にアクセスできます。

filesystem-realm

ファイルシステムが関係する簡単なセキュリティーレルム定義。

重要

filesystem-realm はテクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの「テクノロジプレビュー機能のサポート範囲」を参照してください。

詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with a Filesystem-Based Identity Store」を参照してください。

identity-realm

アイデンティティーが管理モデルで表されるセキュリティーレルム定義。

jdbc-realm

JDBC を使用したデータベースが関係するセキュリティーレルム定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with a Database-Based Identity Store」を参照してください。

key-store-realm

キーストアが関係するセキュリティーレルム定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with Certificates」を参照してください。

ldap-realm

LDAP が関係するセキュリティーレルム定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with a LDAP-Based Identity Store」を参照してください。

properties-realm

プロパティーファイルが関係するセキュリティーレルム定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with a Properties File-Based Identity Store」を参照してください。

token-realm

セキュリティートークンからアイデンティティーを検証および抽出できるセキュリティーレルム定義。

パーミッションマッパー

custom-permission-mapper

カスタムパーミッションマッパーの定義。

logical-permission-mapper

論理パーミッションマッパーの定義。

simple-permission-mapper

簡単に設定されたパーミッションマッパーの定義。

constant-permission-mapper

同じ定数を常に返すパーミッションマッパーの定義。

ロールデコーダー

custom-role-decoder

カスタム RoleDecoder の定義。

simple-role-decoder

単一の属性を取り、それを直接ロールへマップする簡単な RoleDecoder の定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with a Filesystem-Based Identity Store」を参照してください。

ロールマッパー

add-prefix-role-mapper

接頭辞を追加するロールマッパーのロールマッパー定義。

add-suffix-role-mapper

接尾辞を追加するロールマッパーのロールマッパー定義。

aggregate-role-mapper

ロールマッパーが他のロールマッパーの集約であるロールマッパー定義。

constant-role-mapper

ロールの定数セットが常に返されるロールマッパー定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with Certificates」を参照してください。

custom-role-mapper

カスタムロールマッパーの定義。

logical-role-mapper

参照したロールマッパーを 2 つ使用して論理操作を実行するロールマッパーのロールマッパー定義。

SSL コンポーネント

client-ssl-context

接続のクライアント側で使用する SSLContext。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Using a client-ssl-context」を参照してください。

filtering-key-store

`key-store` をフィルターしてキーストアを提供するフィルタリングキー定義。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Using a filtering-key-store」を参照してください。

key-manager

SSLContext の作成に使用されるキーマネージャーリストを作成するためのキーマネージャー定義。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Enable One-way SSL/TLS for the Management Interfaces Using the Elytron Subsystem」を参照してください。

key-store

キーストア定義。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Enable One-way SSL/TLS for the Management Interfaces Using the Elytron Subsystem」を参照してください。

ldap-key-store

LDAP サーバーからキーストアをロードする LDAP キーストア定義。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Using an ldap-key-store」を参照してください。

server-ssl-context

接続のサーバー側で使用する SSL コンテキスト。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Enable One-way SSL/TLS for the Management Interfaces Using the Elytron Subsystem」を参照してください。

trust-manager

SSLContext の作成に使用される `TrustManager` を作成するためのトラストマネージャー定義。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Enable Two-way SSL/TLS for the Management Interfaces using the Elytron Subsystem」を参照してください。

その他

aggregate-providers

2 つ以上の `provider-loader` リソースの集約。

authentication-configuration

リモート接続の確立時、認証用に JBoss EAP およびその他のリソースにデプロイされたクライアントによって使用される個別の認証設定定義。

authentication-context

JBoss EAP およびその他のリソースにデプロイされたクライアントがリモート接続を確立するときに **ssl-context** および **authentication-configuration** を提供するために使用される個別の認証コンテキスト定義。

credential-store

外部サービスのパスワードなどの機密性の高い情報のエイリアスを保持するクレデンシャルストア。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Create a Credential Store 」を参照してください。

dir-context

ディレクトリー (LDAP) サーバーに接続する設定。
詳細は、JBoss EAP『**How to Configure Server Security** 』の「Using an ldap-key-store」を参照してください。

provider-loader

プロバイダーローダーの定義。

security-domain

セキュリティードメイン定義。
詳細は、JBoss EAP『**How to Configure Identity Management** 』の「Configure Authentication with Certificates」を参照してください。

security-property

設定するセキュリティープロパティーの定義。

2.7. コア管理認証

コア管理認証は、**ManagementRealm** を使用してコア管理機能向けに管理インターフェース (HTTP およびネイティブ) をセキュア化します。コア管理認証はコア管理に内蔵され、デフォルトではコアサービスとして有効化および設定されます。管理インターフェースのセキュア化のみを行います。

2.7.1. セキュリティーレルム

セキュリティーレルムは、ユーザー、パスワード、およびグループメンバーシップ情報のアイデンティティーストアで、EJB、web アプリケーション、および管理インターフェースのユーザーを認証するときに使用されます。デフォルトでは、JBoss EAP には事前設定された **ManagementRealm** および **ApplicationRealm** の 2 つのセキュリティーレルムが含まれています。これら両方のセキュリティーレルムはファイルシステムを使用して、ユーザーとパスワード間およびユーザーとグループメンバーシップ情報間のマッピングを保存します。両方のセキュリティーレルムはデフォルトでダイジェスト認証を使用します。

ダイジェストは認証のメカニズムで、プロパティーファイルをマッピングするユーザー名およびパスワードに格納された情報など、さまざまな情報で構成される 1 度限りの一方向ハッシュを使用してユーザーを認証します。これにより、JBoss EAP はネットワーク上でプレーンテキストのパスワードを送信せずに認証を行うことができます。

JBoss EAP インストールには、管理者が両方のレルムにユーザーを追加できるようにするスクリプトが含まれています。ユーザーがこのスクリプトで追加されると、ユーザー名とハッシュ化されたパスワードは対応するユーザー名とパスワードのプロパティーファイルに格納されます。ユーザーが認証を試みると、JBoss EAP は nonce という 1 度限りの番号をクライアントに送信します。クライアントはユーザー名、パスワード、nonce、およびその他のフィールドを使用して一方向ハッシュを生成し、ユー

ザー名、nonce、および一方向ハッシュを JBoss EAP に送信します。JBoss EAP はユーザーのハッシュ化前のパスワードを検索し、提供されたユーザー名、nonce、およびその他のフィールドとともに使用して別の一方向ハッシュを同様に生成します。正しいパスワードなど、すべて同じ情報が両側で使用された場合、ハッシュが一致してユーザーが認証されます。

セキュリティーレルムはデフォルトでダイジェスト認証を使用しますが、他の認証方法を使用するように再設定することもできます。管理インターフェースは起動時に、**ManagementRealm** に設定された認証方法を基にして、有効化される認証方法を判断します。

セキュリティーレルムは承認には全く関与しません。しかし、後で承認の決定に使用されるユーザーのグループメンバーシップ情報をロードするよう設定することが可能です。ユーザーの認証後、ユーザー名を基にグループメンバーシップ情報をロードする 2 つ目の手順が発生します。

デフォルトでは、**ManagementRealm** は管理インターフェースの認証および承認中に使用されます。**ApplicationRealm** は、webアプリケーションと EJB がユーザーの認証および承認時に使用できるデフォルトのレルムです。

2.7.2. デフォルトのセキュリティー

デフォルトでは、コア管理認証は HTTP と ネイティブの各管理インターフェースを、ローカルクライアントとリモートクライアントの 2 つの形式でセキュア化します。これら両方は、デフォルトでは **ManagementRealm** セキュリティーレルムを使用して設定されます。デフォルトの設定は変更可能で、設定を完全に置き換えることもできます。

注記

初期状態の管理インターフェースは、ロールを使用しない簡単なアクセス制御を使用するよう設定されています。デフォルトでは、簡単なアクセス制御を使用するとすべてのユーザーに SuperUser ロールと同じ権限が与えられ、原則的にアクセスが制限されません。

2.7.2.1. ネイティブインターフェースによるローカルおよびリモートクライアント認証

ネイティブインターフェース (管理 CLI) は、稼働中の JBoss EAP インスタンスと同じホストにてローカルで呼び出すことができ、別のマシンからリモートで呼び出すこともできます。ネイティブインターフェースを使用して接続を確立しようとすると、JBoss EAP は **local jboss user** やベーシック認証などの利用可能な SASL 認証のリストをクライアントに提示します。クライアントは希望する認証方法を選択し、JBoss EAP インスタンスとの認証を試行します。認証に失敗すると、他の方法で認証を再試行するか、接続の確立を停止します。ローカルクライアントは **local jboss user** 認証を使用することもできます。これは、ローカルファイルシステムにアクセスできるクライアントの機能を基にしたセキュリティーメカニズムです。これは、ログインしようとしているユーザーが JBoss EAP インスタンスと同じホスト上のローカルファイルシステムにアクセスできる権限があるかを検証します。

この認証は 4 つのステップで行われます。

1. クライアントは、**local jboss user** を使用した認証リクエストが含まれるメッセージをサーバーに送信します。
2. サーバーは 1 度限りのトークンを生成して、固有のファイルに書き込み、そのファイルのフルパスが含まれるメッセージをクライアントに送信します。
3. クライアントはトークンをファイルから読み取って、サーバーに送信し、ローカルでファイルシステムにアクセスできることを確認します。
4. サーバーはトークンを検証し、ファイルを削除します。

このような認証は、ファイルシステムへ物理的にアクセスできれば他のセキュリティーメカニズムは不要であるという原理に基づいています。これは、ユーザーがローカルのファイルシステムにアクセスできれば、ユーザーは新規ユーザーを作成できる権限があり、そうでなければ他のセキュリティーメカニズムが阻止されるという論理です。ローカルユーザーはユーザー名やパスワードの認証なしで管理 CLI にアクセスできるため、サイレント認証と呼ばれることもあります。

この機能は利便性のためと、管理 CLI スクリプトを実行しているローカルユーザーの追加認証を不要にするために有効になっています。通常、ユーザーがローカル設定にアクセスできれば、独自のユーザー詳細を追加することもできるため便利な機能と見なされます。

リモートクライアントとして、別のサーバーや同じサーバーからネイティブインターフェースにアクセスすることもできます。リモートクライアントとしてネイティブインターフェースにアクセスする場合、**`local jboss user`** を使用してクライアントを認証することはできず、ダイジェストなどの別の認証を使用するよう強制されます。**`local jboss user`** を使用したローカルクライアントの認証に失敗すると、自動的にフォールバックされ、リモートクライアントとして別の認証方法の使用を試みます。

注記

ネイティブインターフェースではなく HTTP インターフェースを使用して、別のサーバーや同じサーバーから管理 CLI を呼び出すことができます。CLI であるかに関わらず、すべての HTTP 接続はリモートとして考慮され、ローカルインターフェースの認証では処理されません。

重要

デフォルトでは、ネイティブインターフェースは処理されず、すべての管理 CLI トラフィックは HTTP インターフェースによって処理されます。JBoss EAP 7 は HTTP アップグレードをサポートします。HTTP アップグレードによって、クライアントは HTTP 上で最初の接続を確立できますが、その接続を別のプロトコルにアップグレードするため、リクエストを送信します。管理 CLI の場合、HTTP 上から HTTP インターフェースへの最初のリクエストが作成されますが、接続はネイティブプロトコルにアップグレードされます。この接続は HTTP インターフェース上で処理されますが、通信には HTTP ではなくネイティブプロトコルが使用されます。この代わりに、必要であればネイティブインターフェースを有効化し、使用することができます。

2.7.2.2. HTTP インターフェースによるローカルおよびリモートクライアント認証

HTTP インターフェースは、稼働中の JBoss EAP インスタンスと同じホスト上のクライアントがローカルで呼び出すことができ、他のマシンからクライアントがリモードで呼び出すこともできます。ローカルおよびリモートクライアントは HTTP インターフェースにアクセスできますが、HTTP インターフェースにアクセスするすべてのクライアントはリモート接続として処理されます。

クライアントが HTTP 管理インターフェースへの接続を試みると、JBoss EAP は HTTP 応答とともに状態コード **`401 Unauthorized`** と、ダイジェストや GSSAPI などのサポートされる認証をリストする一連のヘッダーを返信します。ダイジェストのヘッダーには、JBoss EAP によって生成された nonce も含まれます。クライアントはヘッダーを確認して使用する認証方法を選択し、適切な返答を送信します。クライアントがダイジェストを選択した場合、ユーザーにユーザー名とパスワードの入力を要求します。クライアントは、ユーザー名やパスワードなどの提供されたフィールド、nonce、その他の情報を使用し、一方向ハッシュを生成します。その後、一方向ハッシュ、ユーザー名、および nonce を返答として JBoss EAP に返信します。JBoss EAP はその情報を取得して別の一方向ハッシュを生成し、両方のハッシュを比較した結果を基にしてユーザーを認証します。

2.7.3. 高度なセキュリティー

管理インターフェースや認証および承認方法のデフォルト設定を変更する方法は複数あります。

2.7.3.1. 管理インターフェースの更新

JBoss EAP では、管理者は認証および承認方法を変更できるだけでなく、管理インターフェース自体の設定も更新できます。これには複数のオプションがあります。

一方向 SSL/TLS を使用するための管理インターフェースの設定

一方向 SSL/TLS のみを使用して通信を行うよう JBoss EAP 管理コンソールを設定すると、セキュリティーが強化されます。クライアントと管理コンソール間のネットワークトラフィックはすべて暗号化されるため、仲介者攻撃などのセキュリティー攻撃のリスクが軽減されます。JBoss EAP インスタンスの管理者は、そのインスタンスに対して非特権ユーザーよりも多くのパーミッションを持ち、一方向 SSL/TLS を使用するとそのインスタンスの整合性や可用性の保護に役立ちます。JBoss EAP で一方向 SSL/TLS を設定するとき、認証局が署名した証明書は信頼チェーンを提供するため、自己署名証明書よりも優先されます。自己署名証明書は許可されますが、推奨されません。

双方向 SSL/TLS の使用

クライアント認証とも呼ばれる双方向 SSL/TLS 認証は、SSL/TLS 証明書を使用してクライアントとサーバーを認証します。そのため、サーバーの伝えるアイデンティティーだけでなく、クライアントの伝えるアイデンティティーも信頼できます。

新しいセキュリティーレルムの更新および作成

デフォルトのセキュリティーレルムは更新可能で、新しいセキュリティーレルムに置き換えることもできます。

2.7.3.2. アウトバウンド接続の追加

セキュリティーレルムには、LDAP サーバーなどの外部インターフェースに接続するものあります。アウトバウンド接続は、このような接続の確立方法を定義します。事前定義された接続タイプ **`ldap-connection`** は、LDAP サーバーへ接続し、クレデンシャルを検証するために必要な属性と任意の属性をすべて設定します。

2.7.3.3. 管理インターフェースへの RBAC の追加

デフォルトでは、RBAC システムは無効になっています。有効にするには、**`provider`** 属性を **`simple`** から **`rbac`** に変更します。これは、管理 CLI を使用して変更できます。稼働中のサーバーで RBAC を無効化または有効化した場合、サーバー設定をリロードして変更を反映する必要があります。

管理インターフェースに対して RBAC が有効化された場合、アクセスできるリソースとリソースの属性で実行できる操作は、ユーザーに割り当てられたロールによって決定されます。**`Administrator`** または **`SuperUser`** ロールのユーザーのみがアクセス制御システムを閲覧および変更できます。

警告

ユーザーとロールを適切に設定せずに RBAC を有効にすると、管理者が管理インターフェースにログインできなくなることがあります。

管理コンソールでの RBAC の影響

管理コンソールでは、ユーザーに割り当てられたパーミッションに応じて一部の制御と表示が無効になっていることがあります。このような制御と表示は灰色で表示されるか全く表示されません。

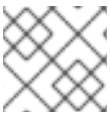
ユーザーがリソース属性の読み取り権限を持たない場合、その属性はコンソールでは空欄で表示されます。たとえば、ほとんどのロールはデータソースのユーザー名およびパスワードフィールドを読み取りできません。

ユーザーがリソース属性の読み取り権限を持ち、書き込み権限は持たない場合、その属性はリソースの編集フォームで無効化されます。ユーザーがリソースへの書き込み権限を持たない場合、そのリソースの編集ボタンは表示されません。

ユーザーがリソースまたは属性へのアクセス権を持たず、そのロールに対してアドレス指定できない場合、そのユーザーのコンソールには表示されません。この例の 1 つがアクセス制御システム自体で、デフォルトでは特定のロールのみが閲覧できます。

管理コンソールは、以下の一般的な RBAC タスクに対してもインターフェースを提供します。

- 各ユーザーに割り当てられたロール、または各ユーザーから除外されたロールの表示および設定。
- 各グループに割り当てられたロール、または各グループから除外されたロールの表示および設定。
- ロールごとのグループおよびユーザーメンバーシップの表示。
- ロールごとのデフォルトメンバーシップの設定。
- スコープ指定されたロールの作成。



注記

現時点では、管理コンソールでは制約を設定できません。

管理 CLI または管理 API での RBAC の影響

RBAC が有効である場合、管理 CLI や管理 API の動作が若干異なります。

読み取りできないリソースや属性は、結果からフィルター処理されます。ロールがフィルターされたリソースや属性をアドレス指定できる場合、結果の **response-headers** セクションにある **filtered-attributes** としてリストされます。ロールがリソースや属性をアドレス指定できない場合はリストされません。

アドレス指定できないリソースにアクセスしようとすると、**Resource Not Found** エラーが発生します。

適切な書き込みまたは読み取り権限のないユーザーが、アドレス指定可能なリソースを読み取りまたは書き込みしようとすると、**Permission Denied** エラーが返されます。

管理 CLI は、管理コンソールと同じ RBAC タスクをすべて実行でき、さらに以下のような追加のタスクも実行できます。

- RBAC の有効化および無効化。
- パーミッション組み合わせポリシーの変更
- アプリケーションリソースおよびリソース機密性制約の設定

JMX 管理 Bean での RBAC の影響

ロールベースのアクセス制御は、3 つの方法で JMX に適用されます。

1. JBoss EAP の管理 API は JMX 管理 Bean として公開されます。JMX 管理 Bean は **core mbeans** と呼ばれ、これらの Bean へのアクセスは基礎の管理 API 自体と全く同じように制御およびフィルターされます。

2. **jmx** サブシステムは、書き込み権限が機密である状態で設定されます。そのため、**Administrator** および **SuperUser** ロールのユーザーのみがこのサブシステムを変更できます。**Auditor** ロールのユーザーはこのサブシステムの設定を読み取ることができます。

3. デフォルトでは、デプロイされたアプリケーションやサービス、またはコアでない MBean によって登録された管理 Bean にはすべての管理ユーザーがアクセスできますが、**Maintainer**、**Operator**、**Administrator** および **SuperUser** ロールを持つユーザーのみが書き込みできます。

RBAC 認証

RBAC は、JBoss EAP と含まれる標準の認証プロバイダーと動作します。

ユーザー名/パスワード

ローカルプロパティーファイルまたは LDAP を使用できる **ManagementRealm** の設定に応じて検証されるユーザーとパスワードの組み合わせを使用して、ユーザーは認証されます。

クライアント証明書

トラストストアはクライアント証明書の認証情報を提供します。

local jboss user

サーバーが同じマシン上で稼働している場合、**jboss-cli** スクリプトは **local jboss user** として自動的に認証を行います。デフォルトでは、**local jboss user** は **SuperUser** グループのメンバーです。

使用されるプロバイダーに関係なく、JBoss EAP はユーザーへロールを割り当てます。**ManagementRealm** または LDAP サーバーで認証を行う場合、これらのシステムはユーザーグループ情報を提供できます。JBoss EAP はこの情報を使用してユーザーにロールを割り当てることもできます。

2.7.3.4. LDAP と管理インターフェースの使用

JBoss EAP には、LDAP サーバーを web および EJB アプリケーションの認証および承認機関として使用できるようにする複数の認証および承認モジュールが含まれています。

LDAP サーバーを管理コンソール、管理 CLI、または管理 API の認証ソースとして使用するには、以下のタスクを実行する必要があります。

1. LDAP サーバーへアウトバウンド接続を作成します。

2. LDAP が有効なセキュリティーレルムを作成するか、既存のセキュリティーレルムが LDAP を使用するよう更新します。

3. 管理インターフェースの新しいセキュリティーレルムを参照します。

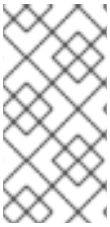
LDAP オーセンティケーターは、最初にリモートディレクトリーサーバーへ接続を確立します。その後、ユーザーが認証システムに渡すユーザー名を使用して検索を実行し、LDAP レコードの完全修飾識別名 (DN) を探します。ユーザーによって提供されるクレデンシャルおよびパスワードとしてユーザー

の DN を使用して、LDAP サーバーへの新しい接続が確立されます。LDAP サーバーの認証に成功すると、DN が有効であることが検証されます。

LDAP が有効なセキュリティーレルムが作成されると、管理インターフェースによる参照が可能になります。管理インターフェースはセキュリティーレルムを認証に使用します。管理インターフェースおよび管理 CLI で認証に双方向 SSL/TLS を使用すると、LDAP サーバーへのアウトバウンド接続を使用するよう JBoss EAP を設定することもできます。

2.7.3.5. JAAS および管理インターフェース

JAAS を使用して管理インターフェースをセキュア化できます。管理インターフェースに JAAS を使用する場合、セキュリティーレルムがセキュリティードメインを使用するように設定する必要があります。これにより、コアサービスとサブシステムの依存関係が発生します。SSL/TLS は、管理インターフェースのセキュア化に JAAS を使用する必要はありませんが、管理者は SSL/TLS を有効にして、セキュアでない状態で機密情報を誤送信しないようにすることが推奨されます。



注記

JBoss EAP インスタンスが **admin-only** モードで実行されている場合、JAAS を使用した管理インターフェースのセキュア化はサポートされません。**admin-only** モードの詳細は、JBoss EAP『**設定ガイド**』の「JBoss EAP の管理専用モードでの実行」を参照してください。

2.8. SECURITY サブシステム

security サブシステムは JAAS API をベースとし、アプリケーションのセキュリティーインフラストラクチャーを提供します。このサブシステムは現在のリクエストに関連するセキュリティーコンテキストを使用して、認証マネージャー、承認マネージャー、監査マネージャー、およびマッピングマネージャーの性能を関係のあるコンテナーに公開します。

認証および承認マネージャーは認証および承認を処理します。マッピングマネージャーは、情報をアプリケーションに渡す前に、プリンシパル、ロール、または属性に対してその情報を追加、変更、または削除します。監査マネージャーは、ユーザーがプリバイダーモジュールを設定して、セキュリティーイベントの報告方法を制御できるようにします。

ほとんどの場合、**security** サブシステムの設定の更新では、管理者はセキュリティードメインの設定のみに集中する必要があります。セキュリティードメインの外部では、変更が必要になる場合があるセキュリティー要素は **deep-copy-subject-mode** のみです。ディープコピーサブジェクトモードに関する詳細は、セキュリティー管理を参照してください。

2.8.1. セキュリティードメイン

セキュリティードメインは JAAS (Java Authentication and Authorization Service) の宣言的なセキュリティー設定で、認証、承認、監査、およびマッピングを制御するために 1 つ以上のアプリケーションによって使用されます。デフォルトでは、**jboss-ejb-policy**、**jboss-web-policy**、**other**、および **jaspitest** の 4 つのセキュリティードメインが含まれます。**jboss-ejb-policy** および **jboss-web-policy** セキュリティードメインは、JBoss EAP インスタンスのデフォルトの承認メカニズムで、アプリケーションの設定済みのセキュリティードメインが認証方法を全く定義しない場合に使用されます。これらのセキュリティードメインと **other** は、認証のために JBoss EAP の内部でも使用され、操作の修正に必要です。**jaspitest** セキュリティードメインは、開発の目的で含まれる簡単な JASPI セキュリティードメインです。

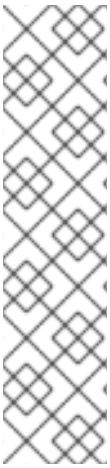
セキュリティードメインは、認証、承認、セキュリティーマッピング、および監査向けの設定で構成されます。セキュリティードメインは、JBoss EAP **security** サブシステムの一部で、ドメインコン

トローラーまたはスタンドアロンサーバーによって一元管理されます。ユーザーは、アプリケーションの要件に応じて必要な数のセキュリティードメインを作成できます。

また、**cache-type** 属性を使用して、セキュリティードメインによって使用される認証キャッシュのタイプを設定することもできます。この属性を削除するとキャッシュは使用されません。このプロパティーに使用できる値は **default** または **infinispan** です。

Elytron セキュリティードメインと PickeBox セキュリティードメインの比較
デプロイメントを単一の Elytron セキュリティードメインか、1 つ以上のレガシー PicketBox セキュリティードメインと関連付ける必要があります。デプロイメントを Elytron と PicketBox の両方に関連付けしてはなりません。これは無効な設定です。

デプロイメントを複数の Elytron セキュリティードメインと関連付けると例外が発生しますが、レガシーセキュリティードメインの場合は複数関連付けることが可能です。



注記

PicketBox の場合、セキュリティードメインは基礎のアイデンティティーストアへのアクセスと承認決定のマッピングの両方をカプセル化します。そのため、PicketBox に複数のストアがある場合は、ソースごとに異なるセキュリティードメインを使用する必要があります。

Elytron では、これら 2 つの機能が分離されています。ストアへのアクセスはセキュリティーレルムで処理され、承認のマッピングはセキュリティードメインによって処理されます。

そのため、独立した PicketBox セキュリティードメインが必要なデプロイメントは、独立した Elytron セキュリティードメインを必ずしも必要としません。

2.8.2. セキュリティーレルムとセキュリティードメインの使用

セキュリティーレルムとセキュリティードメインは、JBoss EAP にデプロイされた web アプリケーションのセキュア化に使用できます。セキュリティーレルムとセキュリティードメインの違いを理解してから、どちらを使用するかを決定することが重要になります。

web アプリケーションと EJB デプロイメントは、セキュリティードメインのみを直接使用できます。セキュリティードメインは、アイデンティティーストアから渡されたアイデンティティー情報を使用するログインモジュールを使用して実際の認証と承認を実行します。アイデンティティー情報にセキュリティーレルムを使用するよう、セキュリティードメインを設定できます。たとえば **other** の場合、アプリケーションは認証と承認情報の取得に使用するセキュリティーレルムを指定できます。セキュリティードメインが外部のアイデンティティーストアを使用するよう設定することも可能です。web アプリケーションと EJB デプロイメントが認証にセキュリティーレルムを直接使用するように設定することはできません。セキュリティードメインは **security** サブシステムの一部でもあり、コアサービスの後にロードされます。

管理インターフェースや EJBが リモーティングエンドポイントなどのコア管理のみがセキュリティーレルムを直接使用できます。セキュリティーレルムは、認証および承認情報を提供するアイデンティティーストアです。コアサービスの 1 つでもあり、サブシステムの起動前にロードされます。**ManagementRealm** および **ApplicationRealm** は初期状態のセキュリティーレルムで、簡単なファイルベースの認証を使用しますが、別の認証を使用するよう設定できます。

2.8.3. セキュリティー監査

セキュリティー監査とは、ログへの書き込みなど、**security** サブシステム内で発生するイベントに対してトリガーするイベントのことです。監査方法は、認証、承認、およびセキュリティーマッピングの

詳細とともにセキュリティードメインの一部として設定されます。セキュリティーイベントの報告方法を制御するため、監査にはプロバイダーモジュールが使用されます。JBoss EAP には複数のセキュリティー監査プロバイダーが含まれますが、カスタムのプロバイダーを使用することもできます。JBoss EAP のコア管理には、**security** サブシステムの一部ではなく別に設定される独自のセキュリティー監査およびロギング機能があります。

2.8.4. セキュリティーマッピング

セキュリティーマッピングを使用すると、認証または承認後、情報がアプリケーションに渡される前に認証情報と承認情報を組み合わせることができます。マップできるのは、承認のロール、認証のプリンシパル、またはプリンシパルやロールでない属性のクレデンシャルです。ロールマッピングは、認証後にロールをサブジェクトに対して追加、置換、または削除するために使用されます。プリンシパルマッピングは、認証後にプリンシパルを変更するために使用されます。クレデンシャルマッピングを使用すると、外部システムの属性をアプリケーションが使用できるように変換することができます。また逆に、クレデンシャルマッピングを使用して、アプリケーションの属性を外部システムが使用できるように変換することもできます。

2.8.5. パスワード vault システム

JBoss EAP には、機密性の高い文字列を暗号化して暗号化されたキーストアに格納し、アプリケーションや検証システムに対して復号化するパスワード vault が含まれています。XML デプロイメント記述子などのプレーンテキストの設定ファイルでは、パスワードや他の機密情報を指定する必要があるときがあります。JBoss EAP のパスワード vault を使用すると、プレーンテキストファイルで使用する機密性の高い文字列をセキュアに格納することができます。

2.8.6. セキュリティードメインの設定

セキュリティードメインは、ドメインコントローラーまたはスタンドアロンサーバーで一元的に設定されます。セキュリティードメインが使用される場合、セキュリティーを別々に設定する代わりに、アプリケーションがセキュリティードメインを使用するよう設定することができます。これにより、ユーザーと管理者は 宣言的セキュリティー を利用することができます。

例

このような設定構造を有効に活用できる一般的な例が、アプリケーションをテスト環境と本番環境の間で移動する処理です。アプリケーションのセキュリティーが個別に設定されている場合、テスト環境から本番環境など、アプリケーションが新しい環境に移されるたびにアプリケーションを更新する必要がある可能性があります。代わりにアプリケーションがセキュリティードメインを使用すると、各環境の JBoss EAP インスタンスのセキュリティードメインは現在の環境に対して適切に設定されます。アプリケーションはコンテナーに依存し、セキュリティードメインを使用して適切なセキュリティー設定を提供できます。

2.8.6.1. ログインモジュール

JBoss EAP には、セキュリティードメイン内で設定されるほとんどのユーザー管理ロールに適する、バンドルされたログインモジュールが複数含まれています。**security** サブシステムは、リレーショナルデータベース、LDAP サーバー、またはフラットファイルからユーザー情報を読み取りできるコアログインモジュールを複数提供します。JBoss EAP はこれらのコアログインモジュールの他に、カスタマイズの必要性に対応する、ユーザー情報や機能を提供するログインモジュールも提供します。

一般的に使用されるログインモジュールの概要

Ldap ログインモジュール

Ldap ログインモジュールは、LDAP サーバーに対して認証を行うログインモジュール実装です **security** サブシステムは **bindDN** を接続情報として使用し、LDAP サーバーに接続します。この

bindDN は、JNDI 初期コンテキストを使用した場合にユーザーおよびロールの**baseCtxDN** および **rolesCtxDN** ツリーを検索する権限があります。ユーザーが認証を試みると、LDAP ログインモジュールは LDAP サーバーへ接続し、ユーザーのクレデンシャルを LDAP サーバーに渡します。認証に成功すると、JBoss EAP 内のそのユーザーに **InitialLDAPContext** が作成され、ユーザーのロールが入力されます。

LdapExtended ログインモジュール

The LdapExtended ログインモジュールは、ユーザーと認証でバインドする関連ロールを検索します。ロールは再帰的にクエリーを行い、DN に従って階層的なロール構造を移動します。ログインモジュールオプションには、JNDI プロバイダーがサポートする指定の LDAP によってオプションがサポートされるかどうかが含まれます。

UsersRoles ログインモジュール

UsersRoles ログインモジュールは、Java プロパティーファイルからロードされる複数のユーザーおよびユーザーロールをサポートする簡単なログインモジュールです。このログインモジュールの主な目的は、アプリケーションとともにデプロイされたプロパティーファイルを使用して複数のユーザーおよびロールのセキュリティー設定を簡単にテストすることです。

Database ログインモジュール

Database ログインモジュールは、認証とロールマッピングをサポートする JDBC (Java Database Connectivity) ベースのログインモジュールです。このログインモジュールは、ユーザー名、パスワード、およびロール情報がリレーショナルデータベースに格納される場合に使用されます。このログインモジュールは、想定される形式のプリンシパルおよびロールが含まれる論理テーブルへの参照を提供して動作します。

Certificate ログインモジュール

Certificate ログインモジュールは、X509 証明書を基にユーザーを認証します。このログインモジュールの典型的なユースケースが、web 層の CLIENT-CERT 認証です。証明書ログインモジュールは認証のみを実行するため、セキュアな web または EJB コンポーネントへのアクセスを完全に定義するには、承認ロールを取得できる他のログインモジュールと組み合わせる必要があります。このログインモジュールの 2 つのサブクラスである **CertRolesLoginModule** および **DatabaseCertLoginModule** は動作を拡張し、プロパティーファイルまたはデータベースから承認ロールを取得します。

Identity ログインモジュール

Identity ログインモジュールは、ハードコードされたユーザー名をモジュールに対して認証されたサブジェクトに関連付ける簡単なログインモジュールです。このモジュールは、プリンシパルのオプションによって指定された名前を使用して **SimplePrincipal** インスタンスを作成します。このログインモジュールは、固定のアイデンティティーをサービスに提供する必要がある場合に便利です。また、指定のプリンシパルに関連するセキュリティーや関連するロールをテストするために、開発環境でも使用できます。

RunAs ログインモジュール

RunAS ログインモジュールは、認証のログインフェーズの間に run-as ロールをスタックにプッシュするヘルパーモジュールです。ログインフェーズ後、コミットまたはアボートフェーズで run-as ロールをスタックからポップします。このログインモジュールの目的は、セキュアな EJB にアクセスするログインモジュールなど、セキュアなリソースにアクセスして認証を実行する必要のあるその他のログインモジュールにロールを提供することです。RunAs ログインモジュールは、run-as ロールの構築が必要なログインモジュールよりも先に設定する必要があります。

Client ログインモジュール

Client ログインモジュールは、呼び出し元のアイデンティティーおよびクレデンシャルの確立時に JBoss クライアントによって使用されるログインモジュールの実装です。新しい **SecurityContext** を作成してプリンシパルとクレデンシャルに割り当て、**SecurityContext** を **ThreadLocal** セキュリティーコンテキストに設定します。Client ログインモジュールは、クライアントが現在のスレッドの呼び出し元を確立するために唯一サポートされるログインモジュールで

す。セキュリティー環境が JBoss EAP **security** サブシステムを使用するよう透過的に設定されていない EJB クライアントとして動作するサーバー環境とスタンドアロンクライアントアプリケーションは、Client ログインモジュールを使用する必要があります。

警告

このログインモジュールは認証を実行しません。サーバー上の後続の認証のために、提供されたログイン情報をサーバー EJB 呼び出しレイヤーにコピーすることもほとんどありません。JBoss EAP 内では、JVM 内の呼び出しに対してユーザーのアイデンティティーを切り替える場合のみサポートされます。リモートクライアントがアイデンティティーを確立する目的ではサポートされません。

SPNEGO ログインモジュール

SPNEGO ログインモジュールは、KDC で呼び出し元のアイデンティティーとクレデンシャルを確立するログインモジュールの実装です。モジュールは JBoss Negotiation プロジェクトの一部で、SPNEGO を実装します。この認証を AdvancedLdap ログインモジュールとチェーンされた設定で使用すると、LDAP サーバーと連携できます。また、このログインモジュールを使用するには、**web** アプリケーションがアプリケーション内で **`NegotiationAuthenticator`** を有効にする必要があります。

RoleMapping ログインモジュール

RoleMapping ログインモジュールは、1 つ以上の宣言的ロールへの認証プロセスの最終結果となるロールのマッピングをサポートします。たとえば、ユーザー John のロールが **`ldapAdmin`** と **`testAdmin`** で、**`web.xml`** または **`ejb-jar.xml`** ファイルで定義されたアクセスの宣言的ロールは **`admin`** であると認証プロセスによって判断された場合、このログインモジュールは**`ldapAdmin`** および **`testAdmin`** ロールを John にマップします。RoleMapping ログインモジュールは、以前マップされたロールのマッピングを変更するため、ログインモジュール設定でオプションのモジュールとして定義する必要があります。

Remoting ログインモジュール

Remoting ログインモジュールは現在認証中のリクエストがリモーティング接続上で受信されたかどうかをチェックします。リクエストがリモーティングインターフェースを使用して受信された場合、リクエストは認証プロセス中に作成されたアイデンティティーに関連付けられます。

RealmDirect ログインモジュール

RealmDirect ログインモジュールは、認証および承認の決定に既存のセキュリティーレルムを使用できるようにします。このモジュールを設定すると、認証の決定とユーザーロールのマッピングに、参照したレルムを使用してアイデンティティー情報を検索します。たとえば、JBoss EAP に同梱される事前設定された **`other`** セキュリティードメインには RealmDirect ログインモジュールがあります。このモジュールに参照されるレルムがない場合、デフォルトで **`ApplicationRealm`** セキュリティーレルムが使用されます。

カスタムモジュール

JBoss EAP セキュリティーフレームワークとバンドルされるログインモジュールがセキュリティー環境の要件に対応できない場合、カスタムログインモジュール実装を作成できます。AuthenticationManager は、Subject プリンシパルの特定の使用パターンを必要とします。AuthenticationManager と動作するログインモジュールを作成するには、JAAS Subject クラスの情報ストレージ機能と、これらの機能の想定される使用方法を完全に理解する必要があります。

UnauthenticatedIdentity ログインモジュールオプションも一般的に使用されます。一部のケースでは、認証された形式でリクエストが受信されないことがあります。UnauthenticatedIdentity

は、**guest** などの特定のアイデンティティーを、関連する認証情報がない状態で作成されたリクエストに割り当てます。これを使用すると、保護されていないサーブレットは特定ロールを必要としない EJB でメソッドを呼び出すことができます。このようなプリンシパルには関連したロールがなく、セキュアでない EJB や、チェックされていないパーミッション制約と関連する EJB メソッドのみにアクセスできます。

2.8.6.2. パスワードスタッキング

スタックでは複数のログインモジュールをチェーンでき、各ログインモジュールは認証中にクレデンシャルの検証とロールの割り当てを提供します。これは多くのユースケースで機能しますが、クレデンシャルの検証とロールの割り当てが複数のユーザー管理ストアに分散されることがあります。

ユーザーは中央の LDAP サーバーで管理され、アプリケーション固有のロールはアプリケーションのリレーショナルデータベースに格納される場合を考えてみましょう。**password-stacking** モジュールオプションはこの関係をキャプチャーします。

パスワードスタッキングを使用するには、各ログインモジュールは、**<module-option>** セクションにある **password-stacking** 属性を **useFirstPass** に設定する必要があります。パスワードスタッキングに設定した以前のモジュールがユーザーを認証した場合、他のすべてのスタッキングモジュールがユーザーによって認証されたこととなり、承認の手順でロールの提供のみを行います。

password-stacking オプションを **useFirstPass** に設定すると、このモジュールは最初にプロパティー名 **javax.security.auth.login.name** で共有されたユーザー名を検索し、**javax.security.auth.login.password** で共有されたパスワードを検索します。

これらのプロパティーが見つかった場合、プリンシパル名とパスワードとして使用されます。見つからなかった場合、プリンシパル名とパスワードはこのログインモジュールによって設定され、プリンシパル名は **javax.security.auth.login.password**、パスワードは **javax.security.auth.login.password** 以下に格納されます。

2.8.6.3. パスワードのハッシュ化

ログインモジュールのほとんどは、クライアントが提供するパスワードをユーザー管理システムに保存されたパスワードと比較する必要があります。通常、これらのモジュールはプレーンテキストのパスワードを使用しますが、プレーンテキストのパスワードがサーバー側に保存されないようにするため、ハッシュ化されたパスワードをサポートするよう設定できます。JBoss EAP は、ハッシュ化するアルゴリズム、エンコーディング、および文字セットを設定する機能をサポートします。さらに、ユーザーパスワードとストアパスワードがハッシュ化されるタイミングも定義します。

重要

Red Hat JBoss Enterprise Application Platform Common Criteria で認定された設定は、SHA-256 よりも弱いハッシュアルゴリズムはサポートしません。

2.8.7. セキュリティー管理

security サブシステムのセキュリティー管理部分は、**security** サブシステムのハイレベルな動作を上書きするために使用されます。各設定は任意になります。ディープコピーサブジェクトモード以外の設定を変更することはあまりありません。

2.8.7.1. ディープコピーモード

ディープコピーサブジェクトモードが無効であるときに (デフォルト設定)、セキュリティーデータ構造をコピーすると、データ構造全体がコピーされずに元のデータ構造への参照が作成されます。この挙

動は効率的ですが、同じアイデンティティーを持つ複数のスレッドによってフラッシュまたはログアウトの操作が行われ、サブジェクトが消去されると、データが破損しやすくなります。

ディープコピーサブジェクトモードが有効になっていると、データ構造の完全コピーと、クローン可能な関連するデータがすべて作成されます。これはよりスレッドセーフになりますが、効率が悪くなります。

2.8.8. その他のコンポーネント

2.8.8.1. JASPI

Java Authentication SPI for Containers (JASPI または JASPIC) は Java アプリケーションのプラグ可能なインターフェースで、 JSR-196 に定義されています。JBoss EAP では JAAS 認証だけでなく、JASPI 認証も使用できます。JASPI 認証はセキュリティードメインのログインモジュールを使用して設定され、これらのモジュールはスタック可能です。**`jaspitest`** セキュリティードメインは、開発用にデフォルトで含まれる簡単な JASPI セキュリティードメインです。web ベースの管理コンソールは JASPI 認証モジュールの設定を公開しません。JBoss EAP にデプロイされたアプリケーションが JASPI セキュリティードメインを使用するには、デプロイメント記述子に特別なオーセンティケーターを設定する必要があります。

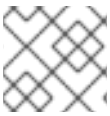
2.8.8.2. JACC

JACC (Java Authorization Contract for Containers) はコンテナーと承認サービスプロバイダー間のコントラクトを定義する規格で、これによりコンテナーによって使用されるプロバイダーが実装されます。これは JSR-115 に定義され、バージョン 1.3 以上ではコア Java EE 仕様の一部となっています。

JBoss EAP は、**`security`** サブシステムのセキュリティー機能内に JACC のサポートを実装します。

2.8.8.3. 粒度の細かい承認および XACML

粒度の細かい承認 (Fine Grained Authorization) は、モジュールへのアクセス権限を付与する意思決定プロセスに関係する要件や変数の変化に管理者が対応できるようにします。そのため、粒度の細かい承認は複雑になります。



注記

JBoss EAP では、web および EJB の XACML バインディングはサポートされません。

JBoss EAP は XACML を媒体として使用し、粒度の細かい承認を実現します。XACML は標準ベースのソリューションを提供し、複雑な粒度の細かい承認に対応します。XACML は、意思決定のポリシー言語やアークテクチャーを定義します。XACML アーキテクチャーには、通常のプログラムフローでリクエストを阻止する PEP (Policy Enforcement Point: ポリシー強制ポイント) が含まれ、PDP (Policy Decision Point: ポリシー決定ポイント) に関連するポリシーを基にアクセスを決定するよう PDP に要求します。PDP は PEP によって作成された XACML リクエストを評価し、ポリシーを確認して以下の 1 つを決定します。

PERMIT
: アクセスは許可されます。

DENY
: アクセスは拒否されます。

INDETERMINATE
: PDP にエラーがあります。

NOTAPPLICABLE

リクエストに足りない属性があるか、一致するポリシーがありません。

XACML には以下の機能もあります。

- Oasis XACML v2.0 ライブラリー
- JAXB v2.0 ベースのオブジェクトモデル
- XACML ポリシーおよび属性を保存および読み出しするための ExistDB 統合

2.8.8.4. SSO

JBoss は **undertow** および **infinispan** サブシステムを使用して、クラスター化された SSO とクラスター化されていない SSO に対して初期状態のサポートを提供します。これには以下の要件があります。

- 認証と承認を処理する設定済みのセキュリティードメイン。
- **SSO** infinispan レプリケーションキャッシュ。これは、管理対象ドメインでは **ha** および **full-ha** プロファイルにあります。スタンドアロンサーバーの場合は**standalone-ha.xml** または **standalone-full-ha.xml** 設定ファイルを使用します。
- **web** キャッシュコンテナーと、その内部の **SSO** レプリケーションキャッシュが存在する必要があります。
- **undertow** サブシステムが SSO を使用するように設定する必要があります。
- SSO 情報を共有する各アプリケーションが同じセキュリティードメインを使用するように設定する必要があります。

第3章 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM における SSO のその他のユースケース

JBoss EAP は、初期状態の機能の他に、その他の SSO のユースケースもサポートします。これらのユースケースには、ブラウザーベースの SSO、デスクトップベースの SSO、およびセキュアトークンサービスを介した SSO に対する SAML が含まれます。

3.1. SAML を使用したブラウザーベースの SSO

ブラウザーベースの SSO の場合、サービスプロバイダーと呼ばれる 1 つ以上の web アプリケーションがハブアンドスポーク型アーキテクチャーの集中管理されたアイデンティティープロバイダーに接続されます。アイデンティティープロバイダー (IDP) は、サービスプロバイダー (スポーク) へのアイデンティティーおよびロール情報の中央のソース (ハブ) として動作します。認証されていないユーザーがサービスプロバイダーの 1 つにアクセスしようとすると、そのユーザーは IDP へリダイレクトされ、認証を行います。IDP がユーザーを認証すると、プリンシパルのロールを指定する SAML トークンを発行し、要求されたサービスプロバイダーにリダイレクトします。SAML トークンは関連するすべてのサービスプロバイダーで使用され、ユーザーのアイデンティティーとアクセス権が判断されます。SSO を使用してサービスプロバイダーで開始されるこのような方法は、サービスプロバイダー開始フロー (Service provider-initiated flow) と呼ばれます。

JBoss EAP は多くの SSO システムと同様に IDP と SP を使用します。これらのコンポーネントは JBoss EAP インスタンス内で実行でき、JBoss EAP の **security** サブシステムとともに動作します。FORM ベースのウェブアプリケーションセキュリティーである SAML v2、HTTP/POST および HTTP/ リダイレクトバインディングも SSO の実行に使用されます。

アイデンティティープロバイダーを作成するには、LDAP やデータベースなどの認証および承認メカニズムを定義する JBoss EAP インスタンスでセキュリティードメイン (例: **idp-domain**) を作成し、アイデンティティーストアとします。**idp-domain** とともに IDP の実行に必要な追加のモジュール (**org.picketlink**) を使用するよう、web アプリケーション (例: **IDP.war**) が設定され、同じ JBoss EAP インスタンスにデプロイされます。 IDP.war はアイデンティティープロバイダーとなります。サービスプロバイダーを作成するため、**SAML2LoginModule** を認証方法として使用するセキュリティードメイン (例: **sp-domain**) が作成されます。web アプリケーション (例: **SP.war**) は追加のモジュール (**org.picketlink**) を使用するよう設定され、**sp-domain** を使用するサービスプロバイダーバルブが含まれます。**SP.war** は **sp-domain** が設定された JBoss EAP インスタンスにデプロイされ、サービスプロバイダーとなります。このプロセスは、**SP2.war** や **SP3.war** などのように 1 つ以上の SP に対してレプリケートでき、1 つ以上の JBoss EAP インスタンス全体でレプリケートできます。

3.1.1. アイデンティティープロバイダー開始フロー (Identity Provider Initiated Flow)

ほとんどの SSO のシナリオでは、SP は有効なアサーションで SAML 応答を SP に送信する IDP に認証リクエストを送信して、フローを開始します。これは SP 開始フロー (SP-initiated flow) と呼ばれます。SAML 2.0 仕様は、IDP 開始フロー (IDP-initiated flow) または未承諾応答フロー (Unsolicited Response flow) と呼ばれる別のフローを定義します。このフローの場合、サービスプロバイダーは認証フローを開始せずに IDP から SAML 応答を受信します。このフローは IDP 側で開始されます。ユーザーは認証されると、リストから特定の SP を選択でき、SP の URL へリダイレクトされます。このフローを有効にするのに特別な設定は必要ありません。

手順

1. ユーザーが IDP にアクセスします。
2. IDP はSAML 要求や応答がないことを確認し、SAML を使用した IDP 開始フローを想定します。

3. IDP はユーザーの認証を行います。

4. 認証後、IDP は、すべての SP アプリケーションにリンクするページをユーザーが取得する、ホストされたセクションを表示します。

5. ユーザーは SP アプリケーションを選択します。

6. IDP は、クエリーパラメーターの SAML 応答に SAML アサーションを持つ SP へユーザーをリダイレクトします。

7. SP は SAML アサーションをチェックし、アクセスを提供します。

3.1.2. グローバルログアウト

1 つの SP で開始されるグローバルログアウトは、IDP およびすべての SP からユーザーをログアウトします。グローバルログアウトの実行後に、SP または IDP のセキュアな部分にアクセスするユーザーは再認証が必要になります。

3.2. デスクトップベースの SSO

デスクトップベースの SSO では、デスクトップ全体でプリンシパルを共有することができ、通常は Active Directory または Kerberos サーバーと、SP である web アプリケーションによって管理されます。この場合、デスクトップ IDP は web アプリケーションの IDP となります。

一般的なセットアップでは、ユーザーは Active Directory ドメインによって管理されるデスクトップへログインします。JBoss Negotiation で設定され、JBoss EAP でホストされる web ブラウザーを介してユーザーは web アプリケーションにアクセスします。web ブラウザーはサインオン情報をユーザーのローカルマシンから web アプリケーションへ転送します。JBoss EAP は、Active Directory または Kerberos サーバーでバックグラウンドの GSS メッセージを使用し、ユーザーを検証します。これにより、ユーザーは web アプリケーションへのシームレスな SSO を実現することができます。

デスクトップベースの SSO を web アプリケーションの IDP としてセットアップするため、IDP サーバーに接続するセキュリティードメインが作成されます。NegotiationAuthenticator は指定の web アプリケーションのバルブとして追加され、JBoss Negotiation は SP コンテナーのクラスパスに追加されます。この代わりに、デスクトップベースの SSO プロバイダーをアイデンティティーストアとして使用して IDP をブラウザーベースの SSO の場合と同様にセットアップすることもできます。

3.3. STS を使用した SSO

JBoss EAP は、STS に接続するためのログインモジュールを複数提供します。STS (**PicketLinkSTS**) として実行することも可能です。**PicketLinkSTS** は他のセキュリティートークンサービスへ複数のインターフェースを定義し、拡張ポイントを提供します。設定を使用して実装をプラグインでき、設定を介してデフォルト値を一部のプロパティーに指定できます。**PicketLinkSTS** はセキュリティートークンを生成および管理しますが、特定タイプのトークンを発行しません。この代わりに、**PicketLinkSTS** は複数のトークンプロバイダーをプラグインできる汎用のインターフェースを定義します。そのため、各トークンタイプのトークンプロバイダーが存在する場合、**PicketLinkSTS** がさまざまなタイプのトークンに対応するよう設定することができます。また、セキュリティートークンのリクエストと応答の形式も指定します。

以下の手順は、JBoss EAP のSTS を使用した場合にセキュリティートークンリクエストが処理される順序を表します。

1. クライアントはセキュリティートークンリクエストを **PicketLinkSTS** に送信します。

2. **PicketLinkSTS** はリクエストメッセージを解析し、JAXB オブジェクトモデルを生成します。

3. **PicketLinkSTS** は設定ファイルを読み取り、必要な場合は **STSConfiguration** オブジェクトを作成します。設定から **WSTrustRequestHandler** への参照を取得し、リクエストの処理をハンドラーインスタンスに委譲します。

4. リクエストがトークンのライフタイム値を指定しない場合など、リクエストハンドラーは必要であれば **STSConfiguration** を使用してデフォルト値を設定します。

5. **WSTrustRequestHandler** は **WSTrustRequestContext** を作成し、**PicketLinkSTS** から受信した JAXB リクエストオブジェクトと呼び出し元プリンシパルを設定します。

6. **WSTrustRequestHandler** は **STSConfiguration** を使用して、リクエストされたトークンタイプを基にリクエストを処理するために使用する必要がある **SecurityTokenProvider** を取得します。これにはプロバイダーが関係し、構築された **WSTrustRequestContext** をパラメーターとして渡します。

7. **SecurityTokenProvider** インスタンスはトークンリクエストを処理し、発行されたトークンをリクエストコンテキストに格納します。

8. **WSTrustRequestHandler** はコンテキストからトークンを取得し、必要な場合は暗号化して、セキュリティートークンが含まれる WS-Trust 応答オブジェクトを構築します。

9. **PicketLinkSTS** はリクエストハンドラーによって生成された応答を書き取り、クライアントへ返します。

STS ログインモジュール (STSIssuingLoginModule、STSValidatingLoginModule、SAML2STSLoginModule など) は、通常はユーザーの認証に STS を使用するよう、セキュリティーセットアップの一部として設定されます。STS はログインモジュールと同じコンテナーに配置するか、web サービス呼び出しや別の技術を使ってリモートでアクセスすることができます。

第4章 ELYTRON サブシステムの例

4.1. 初期状態

JBoss EAP はデフォルトで以下の事前設定されたコンポーネントを提供します。

ApplicationDomain

ApplicationDomain セキュリティードメインは認証に **ApplicationRealm** および **groups-to-roles** を使用します。また、ログインパーミッションの割り当てに**default-permission-mapper** も使用します。

ApplicationRealm

ApplicationRealm セキュリティーレルムは、**application-users.properties** を使用してプリンシパルを認証し、**application-roles.properties** を使用してロールを割り当てるセキュリティーレルムです。これらのファイルは、デフォルトで **EAP_HOME/standalone/configuration** にマップする **jboss.server.config.dir** の下にあります。これらのファイルは、レガシーのデフォルトセキュリティー設定で使用されるファイルと同じです。

application-http-authentication

application-http-authentication http-authentication-factory は HTTP 上の認証に使用できます。これは、**global** provider-http-server-mechanism-factory を使用して認証方法をフィルターし、認証するプリンシパルに **ApplicationDomain** を使用します。**BASIC** および **FORM** 認証を許可し、**BASIC** を **Application Realm** としてアプリケーションに公開します。

application-sasl-authentication

application-sasl-authentication sasl-authentication-factory は SASL を使用した認証に使用できます。これは **configured** sasl-server-factory を使用して認証方法をフィルターし、また **global** provider-sasl-server-factory を使用してプロバイダー名をフィルターします。**application-sasl-authentication** は、プリンシパルの認証に **ApplicationDomain** セキュリティードメインを使用します。

configured (configurable-sasl-server-factory)

これは、メカニズム名を基に使用される **sasl-authentication-factory** をフィルターするために使用されます。この場合、**configured** は **JBOSS-LOCAL-USER** および **DIGEST-MD5** で一致します。また、 **wildfly.sasl.local-user.default-user** を **$local** に設定します。

default-permission-mapper

default-permission-mapper マッパーは、**org.wildfly.security.auth.permission.LoginPermission** を使用してログインパーミッションを割り当て、**org.wildfly.extension.batch.jberet.deployment.BatchPermission** を使用してバッチジョブのパーミッションを割り当てる定数パーミッションマッパーです。バッチパーミッションは **start**、**stop**、**restart**、**abandon**、および **read** で、**javax.batch.operations.JobOperator** と一致します。

elytron (mechanism-provider-filtering-sasl-server-factor)

これは、プロバイダーを基に使用される **sasl-authentication-factory** をフィルターするために使用されます。この場合、**elytron** は **WildFlyElytron** プロバイダー名で一致します。

global (provider-http-server-mechanism-factory)

HTTP 認証ファクトリーの作成時にサポートされる認証方法をリストするために使用される HTTP サーバーファクトリーのメカニズム定義です。

global (provider-sasl-server-factory)

これは、SASL 認証ファクトリーを作成するために使用される SASL サーバーファクトリー定義です。

groups-to-roles

groups-to-roles マッパーは、プリンシパルの **groups** 情報をデコードし、**role** 情報に使用する簡単なロールデコーダーです。

local (マッパー)

local マッパーは、**local** セキュリティーレルムにマップする定数ロールマッパーです。認証を **local** セキュリティーレルムにマップするのに使用されます。

local (セキュリティーレルム)

local セキュリティーレルムは認証を行わず、プリンシパルのアイデンティティーを**$local** に設定します。

ManagementDomain

ManagementDomain セキュリティードメインは認証に 2 つのセキュリティーレルムを使用します。**groups-to-roles** では **ManagementRealm** を使用し、**super-user-mapper** では **local** を使用します。また、ログインパーミッションの割り当てに **default-permission-mapper** も使用します。

ManagementRealm

ManagementRealm セキュリティーレルムは、**mgmt-users.properties** を使用してプリンシパルを認証し、**mgmt-roles.properties** を使用してロールを割り当てるセキュリティーレルムです。これらのファイルは、デフォルトで **EAP_HOME/standalone/configuration** にマップする **jboss.server.config.dir** の下にあります。これらのファイルは、レガシーのデフォルトセキュリティー設定で使用されるファイルと同じです。

management-http-authentication

management-http-authentication http-authentication-factory は、HTTP 上の認証に使用できます。これは **global** provider-http-server-mechanism-factory を使用して認証方法をフィルターし、**ManagementDomain** を認証するプリンシパルに使用します。**DIGEST** 認証を許可し、**ManagementRealm** としてアプリケーションに公開します。

management-sasl-authentication

management-sasl-authentication sasl-authentication-factory は SASL を使用した認証に使用できます。これは **configured** sasl-server-factory を使用して認証方法をフィルターし、また **global** provider-sasl-server-factory を使用してプロバイダー名をフィルターします。**management-sasl-authentication** は、プリンシパルの認証に **ManagementDomain** セキュリティードメインを使用します。また、**local** レルムマッパーを使用して **JBOSS-LOCAL-USER** を使った認証をマップし、**DIGEST-MD5** を使った認証を **ManagementRealm** にマップします。

super-user-mapper

super-user-mapper マッパーは **SuperUser** ロールをプリンシパルにマップする定数ロールマッパーです。

4.1.1. セキュリティー

アプリケーションをセキュアにするため、JBoss EAP には事前設定された HTTP 用の **application-http-authentication** と SASL 用の **application-sasl-authentication** が同梱されています。**application-http-authentication** http-authentication-factory は、認証に **ApplicationRealm** と **groups-to-roles** を使用する **ApplicationDomain** を使用します。**ApplicationRealm** は、ユーザー名、パスワード、およびロール情報に対して**application-users.properties** および **application-roles.properties** が関係する properties-realm です。

管理インターフェースをセキュアにするため、JBoss EAP には事前設定された HTTP 用の **management-http-authentication** と SASL 用の **management-sasl-authentication** が同梱されています。**management-http-authentication** http-authentication-factory は、認証に **ManagementRealm** と **groups-to-roles** を使用する **ManagementDomain** を使用します。**ManagementRealm** は、ユーザー名、パスワード、およびロール情報に対して**mgmt-users.properties** および **mgmt-roles.properties** が関係する properties-realm です。**management-sasl-authentication** sasl-authentication-factory は **JBOSS-LOCAL-USER** 認証に **local** を使用し、**DIGEST-MD5** 認証に **ManagementRealm** を使用します。

4.1.2. 仕組み

デフォルトでは JBoss EAP のユーザーは存在しませんが、この例の目的で以下のユーザーが追加されています。

表4.1 ユーザー

ユーザー名	パスワード	ロール	セキュリティーレルム
Susan	Testing123!		ManagementRealm
Sarah	Testing123!	sample	ApplicationRealm
Frank	Testing123!		ApplicationRealm

JBoss EAP インスタンスは起動時に、4 つの認証ファクトリーとそれらに関連するセキュリティードメイン、セキュリティーレルム、およびその他設定済みのコンポーネントをロードします。

JBOSS-LOCAL-USER を使用して管理 CLI で管理インターフェースにアクセスしようとするユーザーがいる場合 (JBoss EAP インスタンスと同じホストから管理 CLI を実行している)、そのユーザーは **local** セキュリティーレルムを使用して **ManagementDomain** でユーザーを認証しようとする **management-sasl-authentication** sasl-authentication-factory に転送されます。

Susan が異なるホストから管理 CLI を使用して管理インターフェースにアクセスしようとする場合、SASL で **DIGEST-MD5** 認証を使用します。Susan は、 **ManagementRealm** セキュリティーレルムを使用して **ManagementDomain** でユーザーを認証しようとする**management-sasl-authentication** sasl-authentication-factory に転送されます。

Susan が web ベースの管理コンソールを使用して管理インターフェースにアクセスしようとする場合、HTTP で **DIGEST** 認証を使用することになります。Susan は、**ManagementRealm** セキュリティーレルムを使用して **ManagementDomain** でユーザーを認証しようとする**management-http-authentication** http-authentication-factory に転送されます。

アプリケーション **sampleApp1.war** には **/hello.html** と **/secure/hello.html** の 2 つの HTML ファイルがあり、**BASIC** HTTP 認証を使用してパス **/secure/*** をセキュアにします。**Application Realm** を使用し、**sample** ロールが必要です。ユーザーが**sampleApp1.war** にアクセスしようとすると、**application-http-authentication** http-authentication-factory に転送されます。**ApplicationRealm** セキュリティーレルムを使用して **ApplicationDomain** でユーザーを認証しようとします。

Sarah が **/hello.html** を要求すると、認証なしでページを閲覧できます。Sarah が **/secure/hello.html** を要求すると、ユーザー名とパスワードの入力が求められます。ログインに成功した後、Sarah は **/secure/hello.html** を閲覧できます。Frank や Susan を含むすべてのユー

ザーが **/hello.html** にアクセスできますが、Frank と Susan は **/secure/hello.html** にはアクセスできません。Frank は **sample** ロールを持たず、Susan は **ApplicationRealm** セキュリティーレルムに存在しません。

4.2. SSL/TLS を使用した管理インターフェースおよびアプリケーションのセキュア化

ここでは、管理インターフェースとアプリケーションの両方で SSL/TLS に Elytron を使用する場合に JBoss EAP をセキュアにする方法を説明します。

4.2.1. セキュリティー

JBoss EAP では、管理インターフェースとアプリケーションの両方を SSL/TLS でセキュアにすることができます。Elytron を使用してこの設定が統一されたため、管理インターフェースとアプリケーションの両方を同じ SSL/TLS 設定でセキュアにすることができるようになりました。Elytron では、**key-store**、**key-manager**、および **server-ssl-context** を作成して SSL/TLS が設定されます。管理インターフェースに対して SSL/TLS を有効にするには、**http-interface** で **secure-socket-binding** を設定し、**server-ssl-context** を管理インタフェースに割り当てます。これにより、管理インターフェースが HTTP トラフィックに SSL/TLS を使用できるようになります。アプリケーションに対して SSL/TLS を有効にするには、**undertow** サブシステムで **server-ssl-context** を **https-listener** に割り当てます。SSL/TLS の背景情報については、「高度なセキュリティー」を参照してください。

4.2.2. 仕組み

JBoss EAP は起動時に管理インターフェースをコアサービスの一部としてロードします。また、SSL/TLS に対して設定された **http-interface** も起動します。さらに、アプリケーションの SSL/TLS に設定された **undertow** サブシステムと、**server-ssl-context** より SSL/TLS 設定を提供する **elytron** サブシステムも起動します。SSL/TLS が有効になっているセキュアなポート上で管理インターフェースとアプリケーションの両方にアクセスできます。

4.3. 新しいアイデンティティーストアを用いた管理インターフェースおよびアプリケーションのセキュア化

ここでは、JBoss EAP の管理インターフェースおよびアプリケーションの両方を Elytron の新しいアイデンティティーストアでセキュアにする方法を説明します。アプリケーション **sampleApp2.war** は JBoss EAP にデプロイされ、**basicExampleDomain** を使用するよう設定されます。

4.3.1. セキュリティー

JBoss EAP では、**ManagementRealm** と **ApplicationRealm** を超えてアイデンティティーストアで管理インターフェースとアプリケーションをセキュア化できます。Elytron では、同じアイデンティティーストアを使用して管理インターフェースとアプリケーションをセキュアにできますが、個別のアイデンティティーストアを設定することもできます。アイデンティティーストアは、**filesystem-realm**、**jdbc-realm**、**ldap-realm** などのセキュリティーレルムによって表されます。この例のために、**exampleRealm** という名前の **filesystem-realm** が作成されています。また、**exampleDomain** という名前のセキュリティードメインも作成されています。このセキュリティードメインは、**exampleRealm** をアイデンティティーストアとして使用し、**groups-to-roles** ロールマッパーを使用して **exampleRealm** によって提供されるグループ情報をロールにデコードし、マッピングパーミッションに **default-permission-mapper** を使用します。

HTTP 認証では、 **exampleHttpAuthFactory** という **http-authentication-factory** が作成され

ています。これは、認証に **global** HTTP サーバーファクトリーのメカニズムと **exampleDomain** を使用します。また、2 つのメカニズム設定があり、1 つは **basicExampleDomain** として公開される **BASIC** 認証を使用し、もう 1 つは **digestExampleDomain** として公開される **DIGEST** 認証を使用します。HTTP 管理インターフェースは **exampleHttpAuthFactory** を使用するよう設定されています。また、**undertow** サブシステムも **exampleHttpAuthFactory** を使用する新しい **application-security-domain** で設定されています。アプリケーション **sampleApp2.war** は **BASIC** 認証で **basicExampleDomain** を使用するよう設定されています。

SASL 認証では、**exampleSaslAuthFactory** という **sasl-authentication-factory** が作成されています。これは、認証に **configured** SASL サーバーファクトリーと **exampleDomain** を使用します。また、**digestMD5ExampleDomain** として公開される **DIGEST-MD5** 認証が設定されています。管理インターフェースの SASL 設定は **exampleSaslAuthFactory** を使用するよう設定されています。

4.3.2. 仕組み

以下のユーザーが **exampleRealm** に追加されています。

表4.2 exampleRealm ユーザー

ユーザー名	パスワード	ロール
Vincent	samplePass	sample
Issac	samplePass	guest

起動時、JBoss EAP はコアサービスをロードし、**undertow** および **elytron** サブシステムを起動します。これにより、管理インターフェースをセキュアにし、JBoss EAP にデプロイされたアプリケーションに対して **basicExampleDomain**、**digestExampleDomain**、および **digestMD5ExampleDomain** を公開します。

sampleApp2.war がロードされると、**basicExampleDomain** を検索し、セキュアな URL の認証および承認を提供します。**/hello.html** と **/secure/hello.html** の 2 つの HTML ファイルがあり、BASIC 認証を使用してパス **/secure/*** をセキュア化します。セキュアな URL にアクセスするには、**sample** ロールが必要です。

ユーザーの認証時、JBoss EAP へのアクセス方法に応じて、特定のメカニズムを使用してクレデンシャルが提出されます。たとえば、ユーザーが DIGEST 認証で HTTP を使用する管理コンソールにアクセスしようとすると、**digestExampleDomain** として公開される DIGEST 認証を使用して認証されます。BASIC 認証で HTTP を使用する **sampleApp2.war** にアクセスしようとすると、**basicExampleDomain** として公開される BASIC 認証を使用して認証されます。DIGEST 認証で SASL を使用する管理 CLI にアクセスしようとすると、**digestMD5ExampleDomain** として公開される DIGEST-MD5 を使用して認証されます。HTTP 認証は **exampleHttpAuthFactory** を使用し、SASL 認証は **exampleSaslAuthFactory** を使用します。両方の認証ファクトリーは、**exampleDomain** で認証とロールマッピングに対処します。

Vincent または Issac が管理インターフェースにアクセスしようとすると、ユーザー名とパスワードの入力を要求されます。正常にログインした後、各ユーザーは管理操作を実行できます。

Vincent または Issac が **/hello.html** をリクエストすると、そのページを認証なしで閲覧できます。Vincent または Issac が **/secure/hello.html** をリクエストすると、ユーザー名とパスワードの入力を要求されます。正常にログインした後、Vincent は **sample** ロールを持っているため **/secure/hello.html** を閲覧できますが、Issac は **guest** ロールを持っているため

/secure/hello.html を閲覧できません。その他すべてのユーザーはログインせずに**/hello.html** にアクセスできますが、**exampleRealm** に存在するユーザーは Vincent と Issac のみであるため、**/secure/hello.html** にはアクセスできません。

4.4. RBAC を使用した管理インターフェースのセキュア化

ここでは、RBAC と **elytron** サブシステムのアイデンティティーストアを使用して JBoss EAP 管理インターフェースをセキュアにする方法を説明します。

4.4.1. セキュリティー

JBoss EAP では、管理インターフェースで RBAC を使用できます。RBAC の概念については、RBAC の説明を参照してください。この例では、**exampleRealm** というセキュリティーレルムを使用します。ロールデコーダー、セキュリティードメイン、認証ファクトリーなどの残りのセキュリティー設定は、新しいアイデンティティーストアを用いた管理インターフェースおよびアプリケーションのセキュア化 と同じになります。RBAC は、管理インターフェースに対して **provider** 属性を **rbac** に設定し、希望のユーザーとロールで **exampleRealm** を更新すると有効になります。

4.4.2. 仕組み

この例では、以下のユーザーが既存のセキュリティーレルムに追加されています。

表4.3 exampleRealm ユーザー

ユーザー名	パスワード
Suzy	Testing123!
Tim	Testing123!
Emily	Testing123!
Zach	Testing123!
Natalie	Testing123!

グループメンバーシップを基に、ユーザーは以下の RBAC ロールにマップされています。

表4.4 RBAC ロール

ユーザー名	RBAC ロール
Suzy	SuperUser
Tim	Administrator
Emily	Maintainer
Zach	Deployer

ユーザー名	RBAC ロール
Natalie	Monitor

起動時、JBoss EAP はコアサービスと、セキュリティーおよび RBAC 設定をロードする **elytron** サブシステムをロードします。RBAC が有効になっていない場合、**exampleRealm** のユーザーはすべて **SuperUser** として考慮され、アクセスが無制限になります。RBACは有効になっているため、各ユーザーは持っているロールに応じて制限されるようになります。上記の表のとおり、Suzy、Tim、Emily Zach および Natalie は、異なるロールを持っています。たとえば、Suzy と Tim のみがアクセス制御情報の読み取りと編集を行えます。Suzy、Tim、および Emily はランタイム状態とその他の永続設定の情報を編集できます。**Zach** もランタイム状態とその他の永続設定の情報を編集できますが、アプリケーションリソースに関係するもののみに限定されます。Suzy、Tim、Emily、Zack、および Natalie は設定および状態情報を読み取りできますが、Natalie は何も更新できません。各ロールの詳細と JBoss EAP による RBAC の処理方法については、「ロールベースのアクセス制御」と「管理インターフェースへの RBAC の追加」を参照してください。

4.5. KERBEROS を使用した WEB アプリケーションに対する SSO の提供

ここでは、JBoss で Kerberos を使用して web アプリケーションに SSO を提供する方法について説明します。JBoss EAP インスタンス **EAP1** は作成済みで、スタンドアロンサーバーとして実行されています。**sampleAppA** と **sampleAppB** の 2 つのアプリケーションが **EAP1** にデプロイされています。両方のアプリケーションと **EAP1** は、Kerberos でデスクトップベースの SSO を使用して認証するよう設定されています。

4.5.1. セキュリティー

JBoss EAP は、**SPNEGO** 認証を使用した Kerberos での認証を提供します。Kerberos と SPNEGO に特化した情報は、「サードパーティーの SSO 実装」を参照してください。JBoss EAP とデプロイされた web アプリケーションが認証に Kerberos を使用するようにするには、**kerberos-security-factory** を作成して Kerberos サーバーに接続します。Kerberos サーバーからユーザーにロールを割り当てるために、セキュリティーレルム、ロールマッパー、およびセキュリティードメインも作成されます。**kerberos-security-factory** を使用し、認証とロールの割り当てにセキュリティードメインを使用する **http-authentication-factory** が作成されます。認証方法は **SPNEGO** 認証を使用し、**exampleSpnegoDomain** として公開されます。また、**undertow** サブシステムは、認証に **http-authentication-factory** を使用するよう設定されます。

sampleAppA と **sampleAppB** の両方は、認証の実行に **exampleSpnegoDomain** を使用するよう設定され、承認にユーザーのロールを取得します。セキュリティートークンをオペレーティングシステムからブラウザーに渡すことができない場合のために、**FORM** 認証をフォールバック認証として両方のアプリケーションに設定することもできます。**FORM** 認証がフォールバックとして設定された場合、追加の認証方法とサポートするセキュリティードメインを設定する必要があります。認証方法は Kerberos と **SPNEGO** には依存せず、**FORM** 認証のみをサポートする必要があります。この場合、追加の認証とサポートするセキュリティードメインを **FORM** 認証に対して設定し、 **exampleFormDomain** として公開する必要があります。各アプリケーションは **exampleFormDomain** を使用し、**FORM** 認証をフォールバックとして提供するよう、設定されます。また、各アプリケーションはパス **/secure/*** をセキュアにするよう設定され、承認を処理するために独自のロールリストを提供します。

4.5.2. 仕組み

以下のユーザーが Kerberos サーバーに作成されています。

表4.5 Kerberos ユーザー

ユーザー名	パスワード
Sande	samplePass
Andrea	samplePass
Betty	samplePass
Chuck	samplePass

セキュリティードメインを使用して以下のロールがユーザーにマップされます。

表4.6 ユーザーロール

ユーザー名	ロール
Sande	all
Andrea	A
Betty	B
Chuck	

各アプリケーションには以下のロールが設定されています。

表4.7 アプリケーションロール

アプリケーション/SP	許可されるロール
sampleAppA	all、A
sampleAppB	all、B

起動時に、**EAP1** はコアサービスをロードしてから**elytron** およびその他のサブシステムをロードします。**kerberos-security-factory** は Kerberos サーバーへの接続を確立します。**sampleAppA** と **sampleAppB** の両方がデプロイされ、認証のために**exampleSpnegoDomain** と **exampleFormDomain** に接続します。

Sande は Kerberos でセキュア化されたコンピューターにログインしています。Sande はブラウザーを開き、**sampleAppA/secure/hello.html** へのアクセスを試みます。このページはセキュアであるため、認証が必要になります。**EAP1** は、Sande のコンピューターに設定されている Kerberos Key Distribution Center (Kerberos サーバー) へのキーを要求するリクエストを送信するよう、ブラウザーに指示します。ブラウザーがキーを取得した後、**sampleAppA** に送信されます。**sampleAppA** は **exampleSpnegoDomain** を使用してチケットを JBoss EAP に送信します。そこでチケットがアンパックされ、**kerberos-security-factory** の設定済みの Kerberos サーバーとともに認証が実行されます。チケットが認証されたら、Sande のロールは **sampleAppA** に戻され、承認が実行されます。Sande は **all** ロールを持っているため、**sampleAppA/secure/hello.html** にアクセスできます。

Sande が **sampleAppB/secure/hello.html** にアクセスする場合も同じ処理が発生します。**all** ロールを持っているため、アクセスが許可されます。Andrea と Betty にも同じ処理が発生しますが、Andrea は **sampleAppA/secure/hello.html** のみにアクセスでき、**sampleAppB/secure/hello.html** にはアクセスできません。Betty はこの逆で、**sampleAppB/secure/hello.html** のみにアクセスでき、**sampleAppA/secure/hello.html** にはアクセスできません。Chuck は **sampleAppA/secure/hello.html** または **sampleAppB/secure/hello.html** の認証に成功しますが、ロールを持たないためどちらにもアクセスできません。

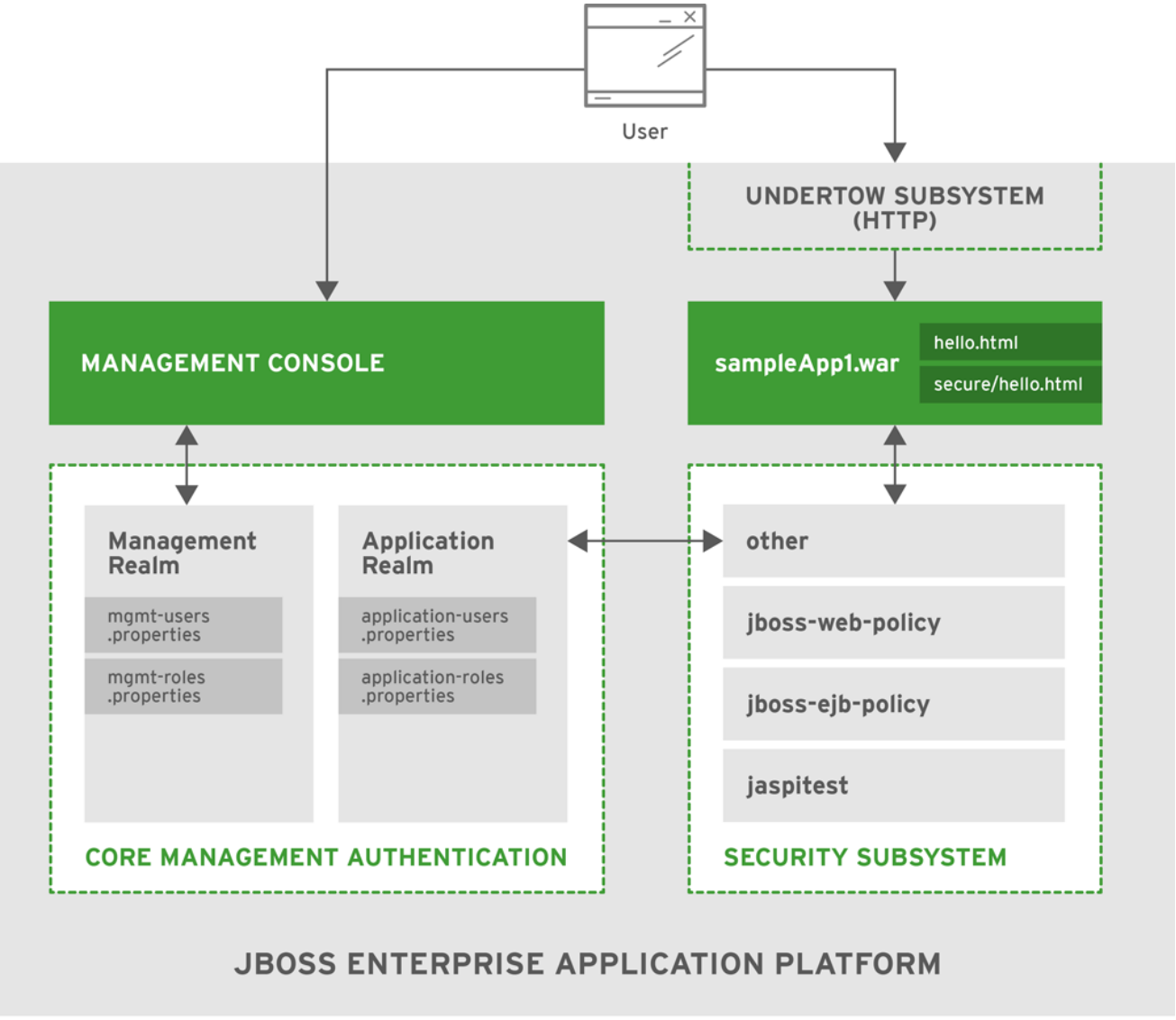
オフィスのネットサークに接続する個人のラップトップなど、Sande が Kerberos でセキュア化されていないコンピューターから **sampleAppA/secure/hello.html** にアクセスしようとすると、フォールバックとして **FORM** ログインページに転送されます。Sande のクレデンシャルはフォールバックの認証を使用して認証され、承認の処理が続行されます。

第5章 レガシーのコア管理とセキュリティーサブシステムの例

JBoss EAP のセキュリティーとコンポーネントがどのように一緒に機能するかを理解する 1 つの方法が、実際のシナリオを検証することです。以下のセクションでは、さまざまな JBoss EAP セキュリティーコンポーネントおよび設定オプションが関係する一般的かつ現実的な状況を複数取り上げます。単一または複数のアプリケーションをセキュア化する方法と、管理インターフェースをセキュア化する方法を中心に説明します。

5.1. 初期状態の RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

ここでは、初期インストール後に設定が変更されなかった場合に JBoss EAP とサンプルアプリケーションをセキュアにする方法を説明します。アプリケーション **`sampleApp1.war`** はデプロイされ、コンテナーベースのセキュリティーを使用するよう設定されています。



JBOSS_397679_0416

5.1.1. 初期設定のコア管理認証

5.1.1.1. セキュリティー

コア管理認証は、**`ManagementRealm`** と **`ApplicationRealm`** の 2 つの事前設定されたセキュリ

ティーレルムをデフォルトで提供します。これらのレルムはプロパティーファイルを使用して、ユーザー名、パスワード、およびロールを格納します。認証情報と管理 API の格納に使用される **ManagementRealm** は、JBoss EAP インスタンスと同じホストで CLI を使用するユーザーに対してローカル認証を定義し、有効にします。**ApplicationRealm** は、管理 API 以外のアプリケーション向けに、認証および承認情報を格納するために事前設定されています。さらに、 **ApplicationRealm** はローカル認証が有効な状態で事前設定されていますが、これは一般的に使用されません。

5.1.1.2. 仕組み

このシナリオでは、JBoss EAP のデフォルトインストールに以下のユーザーが追加されています。

表5.1 ユーザー

ユーザー名	パスワード	ロール	セキュリティーレルム
Susan	Testing123!		ManagementRealm
Sarah	Testing123!	sample	ApplicationRealm
Frank	Testing123!		ApplicationRealm

JBoss EAP インスタンスは、起動時に **ManagementRealm** および **ApplicationRealm** セキュリティーレルムをロードします。

Susan が管理インターフェース、HTTP、または CLI にアクセスしようとすると、認証が必要になります。Susan のユーザー名、パスワード、およびロールが **ManagementRealm** のエントリーと一致する必要があります。デフォルトでは、管理 API にアクセスする必要があるルールはありません。Sarah と Frank は **ManagementRealm** セキュリティーレルムには存在しないため、管理 API にアクセスできません。

5.1.2. 初期設定のセキュリティーサブシステム

5.1.2.1. セキュリティー

security サブシステムには、**other**、**jboss-web-policy**、**jboss-ejb-policy**、および **jaspitest** の 4 つのセキュリティードメインが事前設定されています。**other** セキュリティードメインは、デフォルトで **ApplicationRealm** を使用する、ログインモジュールに指定されたレルムに委譲して認証と承認を実行します。

デフォルトのセキュリティーレルムとデフォルトのセキュリティードメインに関する詳細は、「セキュリティーレルム」と「セキュリティードメイン」を参照してください。

5.1.2.2. 仕組み

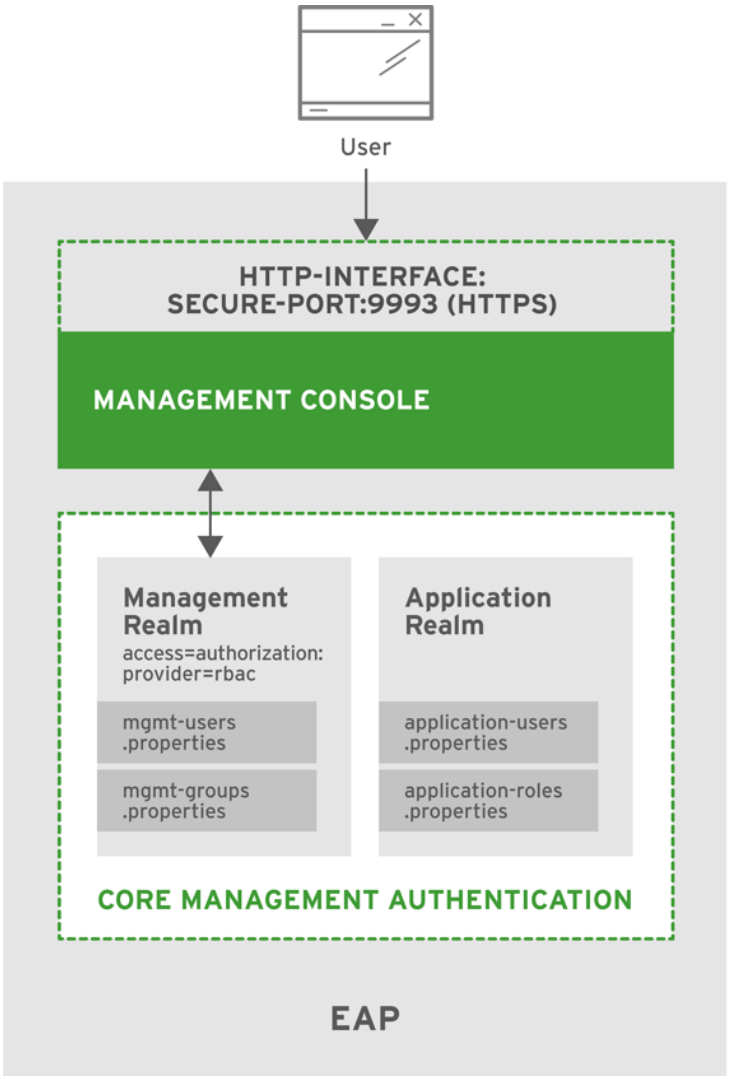
アプリケーション **sampleApp1.war** には **/hello.html** と **/secure/hello.html** の 2 つの HTML ファイルがあり、BASIC HTTP 認証を使用してパス **/secure/*** をセキュア化します。**other** セキュリティードメインを使用し、**sample** ロールを必要とします。**other** セキュリティードメインは認証および承認情報を **ApplicationRealm** に委ねるため、前セクション に記載されている **Users** の表のユーザーを参照します。

Sarah が **/hello.html** を要求すると、認証なしでページを閲覧できます。Sarah が **/secure/hello.html** を要求すると、ユーザー名とパスワードの入力が求められます。ログインに成

功した後、Sarah は **/secure/hello.html** を閲覧できます。Frank や Susan を含むすべてのユーザーが **/hello.html** にアクセスできますが、Frank と Susan は **/secure/hello.html** にはアクセスできません。Frank は **sample** ロールを持たず、Susan は **ApplicationRealm** セキュリティーレルムに存在しません。

5.2. 管理インターフェースに HTTPS と RBAC が追加された RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

ここでは、管理インターフェースで HTTPS が有効で、RBAC が **ManagementRealm** セキュリティーレルムに追加されている場合に JBoss EAP をセキュアにする方法を取り上げます。



5.2.1. セキュリティー

JBoss EAP は管理コンソールを含む、管理インターフェースによる HTTPS の使用をサポートします。HTTPS を有効にするには、**secure-interface** 要素を設定し、**secure-port** を管理インターフェースの **http-interface** セクションに追加し、希望の設定 (サーバー ID、プロトコル、キーストア、エイリアスなど) で既存または新規セキュリティーレルムを設定します。これにより、管理インターフェースがすべての HTTP トラフィックに SSL/TLS を使用できるようになります。HTTPS と管理インターフェースのセキュア化に関する背景情報は、「高度なセキュリティー」を参照してください。

JBoss EAP では、複数の異なる認証スキームを使用して、管理インターフェースの RBAC を有効にすることもできます。この例では、**ManagementRealm** で使用される既存のプロパティーファイルでユーザー名とパスワードを使用します。RBAC を有効にするには、管理インターフェースに対して **provider** 属性を **rbac** に設定し、希望のユーザーとロールで**ManagementRealm** を更新します。

5.2.2. 仕組み

この例では、以下のユーザーが既存のセキュリティーレルムに追加されています。

表5.2 管理ユーザー

ユーザー名	パスワード	セキュリティーレルム
Suzy	Testing123!	ManagementRealm
Tim	Testing123!	ManagementRealm
Emily	Testing123!	ManagementRealm
Zach	Testing123!	ManagementRealm
Natalie	Testing123!	ManagementRealm

グループメンバーシップを基に、ユーザーは以下の RBAC ロールにマップされています。

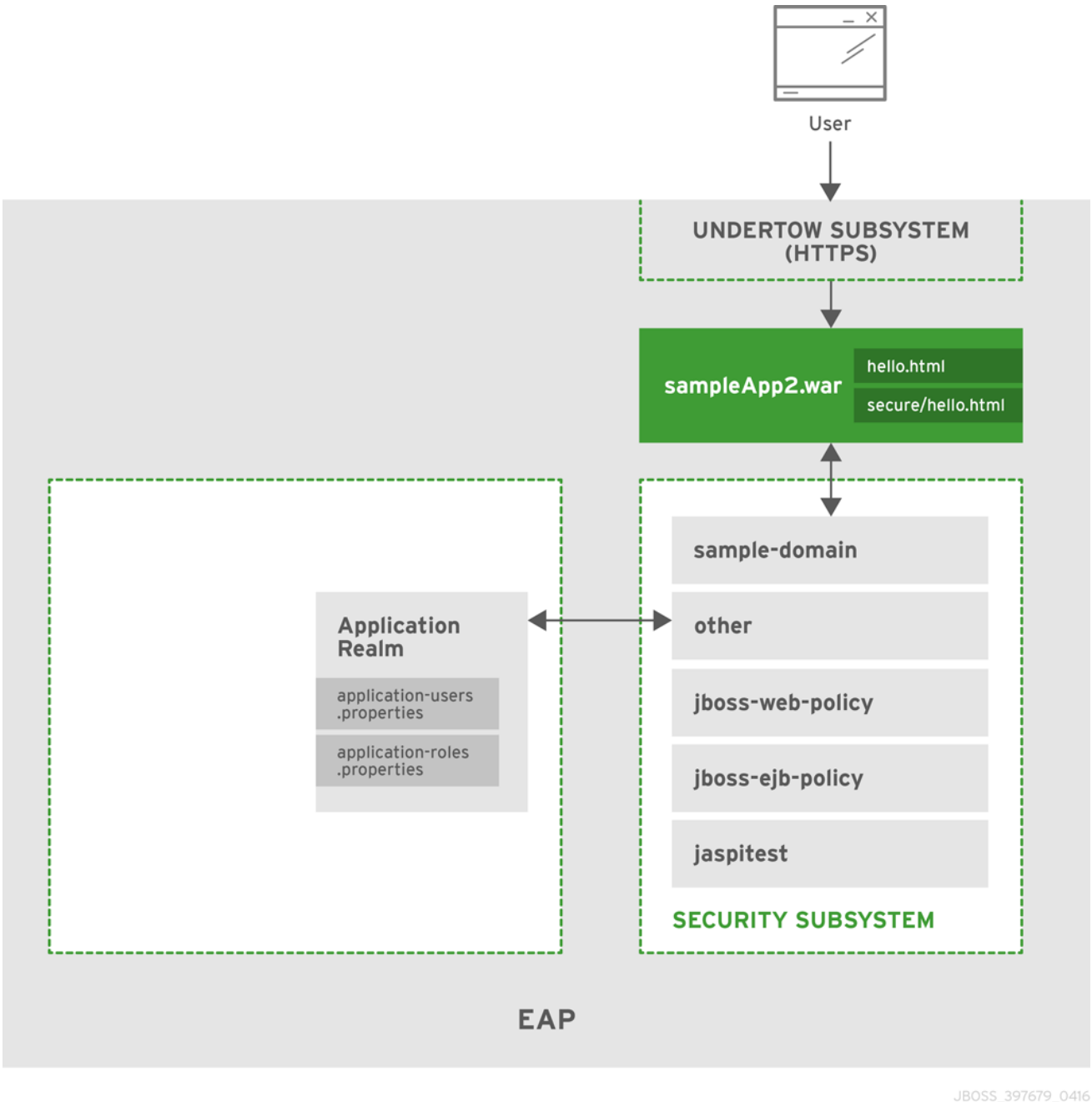
表5.3 RBAC ロール

ユーザー名	RBAC ロール
Suzy	SuperUser
Tim	Administrator
Emily	Maintainer
Zach	Deployer
Natalie	Monitor

起動時、JBoss EAP は **`ManagementRealm`** と RBAC 設定、および管理インターフェースをコアサービスの一部としてロードします。また、管理インターフェースの HTTPS 用に設定された **`http-interface`** も起動されます。ユーザーは HTTPS を介して管理インターフェースにアクセスできるようになり、RBAC も有効化および設定されます。RBAC が有効になっていない場合、**`ManagementRealm`** セキュリティーレルムのユーザーはすべて **`SuperUser`** として考慮され、アクセスが無制限になります。RBAC は有効になっているため、各ユーザーは持っているロールに応じて制限されるようになります。上記の表のとおり、Suzy、Tim、Emily Zach および Natalie は、異なるロールを持っています。たとえば、Suzy と Tim のみがアクセス制御情報の読み取りと編集を行えます。Suzy、Tim、および Emily はランタイム状態とその他の永続設定の情報を編集できます。**`Zach`** もランタイム状態とその他の永続設定の情報を編集できますが、アプリケーションリソース関係のみに限定されます。Suzy、Tim、Emily、Zack、および Natalie は設定および状態情報を読み取りできますが、Natalie は何も更新できません。各ロールの詳細と JBoss EAP による RBAC の処理方法については、「ロールベースのアクセス制御」と「管理インターフェースへの RBAC の追加」を参照してください。

5.3. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM と HTTPS を含む更新された SECURITY サブシステム

ここでは、新しいセキュリティードメインが追加され、HTTPS が有効化された場合に JBoss EAP で実行しているサンプルアプリケーションをセキュアにする方法を取り上げます。アプリケーション **sampleApp2.war** はデプロイされ、コンテナベースのセキュリティー、**sample-domain**、および HTTPS を使用するよう設定されています。



5.3.1. セキュリティー

JBoss EAP は、アプリケーション向けの HTTPS の使用をサポートし、**undertow** サブシステムを使用して処理されます。HTTPS の新しいコネクターは **undertow** サブシステムに追加され、プロトコル、ポート、キーストアなどの希望の設定で設定されます。設定が保存されると、**web** アプリケーションは設定されたポート上で HTTPS トラフィックを許可できるようになります。**sample-domain** という新しいセキュリティードメインが追加され、認証に **IdentityLoginModule** を使用します。**sampleApp2.war** はユーザーの認証に **sample-domain** を使用するよう設定されます。

5.3.2. 仕組み

セキュリティードメイン **sample-domain** が作成され、**IdentityLoginModule** を使用するよう設定されています。以下のクレデンシャルがログインモジュールに設定されています。

表5.4 sample-domain ユーザー

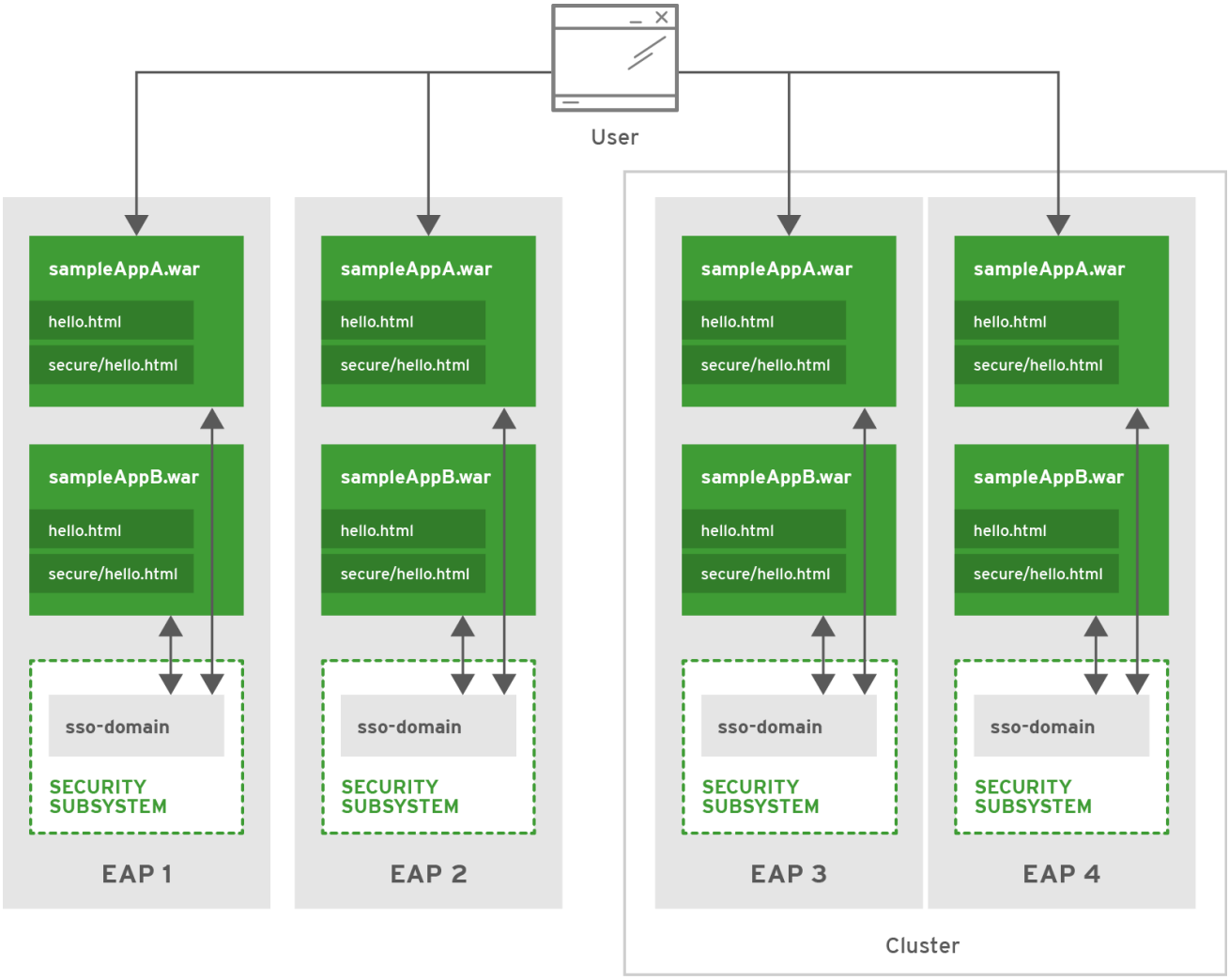
ユーザー名	パスワード	ロール
Vincent	samplePass	sample

JBoss EAP は起動時にコアサービスをロードし、すべての web アプリケーションの HTTPS 接続を管理する **undertow** サブシステムと、**sample-domain** の HTTPS 接続を管理する **security** サブシステムを起動します。**sampleApp2.war** がロードされ、セキュアな URL に認証と承認を提供するための **sample-domain** を検索します。**sampleApp2.war** には **/hello.html** と **/secure/hello.html** の 2 つの HTML ファイルがあり、BASIC HTTP 認証を使用してパス **/secure/*** をセキュア化します。**sample-domain** セキュリティードメインを使用し、**sample** ロールを必要とします。

Vincent が **/hello.html** をリクエストすると、Vincent は認証なしでこのページを閲覧できます。Vincent が **/secure/hello.html** をリクエストすると、ユーザー名とパスワードの入力が要求されます。正常にログインした後、Vincent は **/secure/hello.html** を閲覧できます。その他すべてのユーザーはログインせずに **/hello.html** にアクセスできますが、**sample-domain** に存在するユーザーは Vincent のみであるため、**/secure/hello.html** にはアクセスできません。これは、HTTPS 上で処理されるすべてのトラフィックにも適用されます。

5.4. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM における WEB アプリケーションの SSO

ここでは、JBoss EAP で web アプリケーションがクラスター化された SSO およびクラスター化されていない SSO を使用する方法を取り上げます。**EAP1**、**EAP2**、**EAP3**、および **EAP4** の 4 つの JBoss EAP インスタンスが作成されています。**EAP1** と **EAP2** はスタンドアロンサーバーとして動作し、**EAP3** と **EAP4** は 2 ノードクラスターとして動作します。**sampleAppA** と **sampleAppB** の 2 つの web アプリケーションが各 JBoss EAP インスタンスにデプロイされています。



5.4.1. セキュリティー

JBoss EAP は **security**、**undertow**、および **infinispan** サブシステムの組み合わせを使用し、web アプリケーションでクラスター化された SSO とクラスター化されていない SSO のサポートを提供します。**security** サブシステムは認証と承認を実行するためのセキュリティードメインを提供します。**infinispan** および **undertow** サブシステムは、JBoss EAP インスタンス上または JBoss EAP クラスター全体のすべての web アプリケーション間で SSO 情報をキャッシュおよび分散できるようにします。4 つの EAP インスタンスはすべてセキュリティードメイン **sso-domain** を持ち、**IdentityLoginModule** を使用するよう設定されています。**sampleAppA** および **sampleAppB** は、**sso-domain** セキュリティードメインを使用してパス **/secure/*** をセキュア化するよう設定され、アクセスするには **sample** ロールが必要です。以下のクレデンシャルが**sso-domain** ログインモジュールに設定されています。

表5.5 sso-domain ユーザー

ユーザー名	パスワード	ロール
Jane	samplePass	sample

4 つの JBoss EAP インスタンスはすべて **standalone-full-ha** または **full-ha** プロファイルで起動するよう設定され、このシナリオで SSO を有効化するのに必要な **infinispan** サブシステムとその他の機能を追加します。**web** キャッシュコンテナーとレプリケートされた**SSO** キャッシュも追加さ

れ、**undertow** サブシステムはこれら両方を使用するよう設定されています。**EAP1** と **EAP2** の **undertow** サブシステムはクラスター化されていない SSO に対して設定され、**EAP3** および **EAP4** が含まれるクラスターはクラスター化された SSO を使用するよう設定されています。

アプリケーションのクラスター化とクラスター化された SSO

クラスター化された web アプリケーションとクラスター化された SSO には明らかな違いがあります。クラスター化された web アプリケーションは、そのアプリケーションをホストするための負荷を分散するためにクラスターのノード全体で分散されます。分散可能なクラスター化されたアプリケーションでは、新しいセッションと既存セッションの変更はすべてクラスターの別のメンバーへレプリケートされます。クラスター化された SSO は、アプリケーション自体がクラスター化されているかどうかに関わらず、セキュリティーコンテキストとアイデンティティー情報のレプリケーションを可能にします。これらの技術は一緒に使用することができますが、相互に排他的であり、独立して使用することもできます。

5.4.2. 仕組み

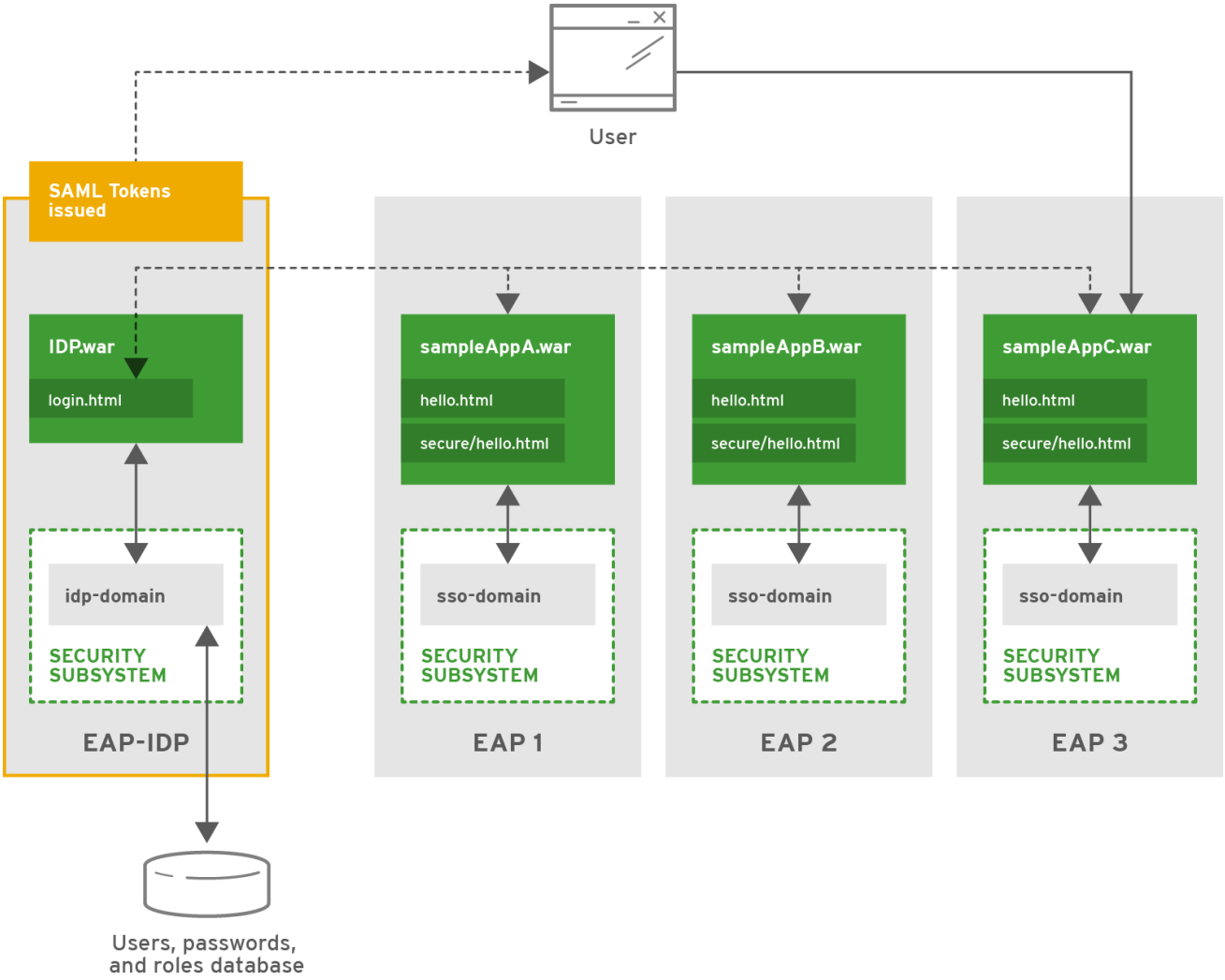
JBoss EAP は起動時にコアサービスをロードし、**sso-domain** と SSO 情報の関連キャッシュを管理する **security**、**undertow**、および **infinispan** サブシステムを起動します。4 つの JBoss EAP インスタンスすべてに **sampleAppA.war** と **sampleAppB.war** がロードされ、各インスタンスは認証と承認を提供するために **sso-domain** を検索します。

Jane が **EAP1/sampleAppA/secure/hello.html** にアクセスしようとすると、認証が要求されます。正しい情報を提供した後、Jane は **EAP1/sampleAppA/secure/hello.html** の閲覧が許可されます。Jane のセッションは **undertow** および **infinispan** サブシステムによって使用される SSO キャッシュに追加されます。Jane が **EAP1/sampleAppB/secure/hello.html** にアクセスしようとすると、再認証は要求されません。**EAP1** で実行されている **sampleAppB** は、**undertow** サブシステムキャッシュと **infinispan** サブシステムを使用して Jane のセッションを検索します。Jane はすでに認証されているため、アクセスを許可します。Jane が **EAP2/sampleAppA/secure/hello.html** または **EAP2/sampleAppB/secure/hello.html** にアクセスしようとすると、再度認証が要求されます。これは、**EAP1** と **EAP2** はキャッシュを共有しないためです。

Jane が **EAP3/sampleAppA/secure/hello.html** にアクセスしようとすると、認証を要求され、Jane のセッションは SSO キャッシュに保存されます。これらのキャッシュはクラスター全体で保存されるため、Jane が **EAP3/sampleAppB/secure/hello.html**、**EAP4/sampleAppA/secure/hello.html**、または **EAP4/sampleAppB/secure/hello.html** にログインする際に再認証する必要ありません。**EAP3** または **EAP4** のいずれかが再起動した場合、他の JBoss EAP インスタンスとクラスターは稼働しているため、Jane の SSO 情報はキャッシュに保持されます。同様に、Jane のセッションが無効化された場合はキャッシュ全体で無効化され、クラスターでアクセスしようとするアプリケーションやサーバーに関係なく、再認証が要求されます。

5.5. SAML でブラウザーベースの SSO を使用する複数の RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM インスタンスと複数のアプリケーション

ここでは、JBoss EAP の複数のインスタンスをセキュアにする方法と、ブラウザーベースの SSO が追加される方法を取り上げます。**EAP1**、**EAP2**、および **EAP3** の個別でクラスター化されていない 3 つの JBoss EAP インスタンスが設定されます。**sampleAppA.war**、**sampleAppB.war**、および **sampleAppC.war** の 3 つのサンプルアプリケーションは、認証にブラウザーベースの SSO を使用するよう設定されます。



5.5.1. セキュリティー

JBoss EAP は、web アプリケーションで SAML を介してブラウザーベースの SSO をサポートし、アイデンティティープロバイダーのホストもサポートします。アイデンティティープロバイダーをホストするには、セキュリティードメイン (この例では **idp-domain**) に定義した認証方法を設定する必要があります。この例では、 **idp-domain** が **DatabaseLoginModule** を認証方法として使用するよう設定されます。IDP アプリケーション (例: **IDP.war**) がその JBoss EAP インスタンス (この例では **EAP-IDP**) にデプロイされ、**idp-domain** をアイデンティティーストアとして使用するよう設定されます。**IDP.war** は、アプリケーションの機能とともにアイデンティティーストアを使用してユーザーを認証し、ユーザーのアイデンティティーとロール情報が含まれる SAML トークンを発行します。**EAP1**、**EAP2**、および **EAP3** の 3 つの JBoss EAP インスタンスは、それぞれ個別の SP として動作するアプリケーション **sampleAppA.war**、**sampleAppB.war**、および **sampleAppC** をホストします。各 JBoss EAP インスタンスには、**SAML2LoginModule** を認証方法として使用するよう設定されている **sso-domain** セキュリティードメインがあります。各アプリケーションには、SSO をサポートし、 **IDP.war** をアイデンティティープロバイダーとして使用し、認証に HTTP/POST バインディングを使用する機能が含まれています。また、各アプリケーションは **sso-domain** を使用してパス **/secure/*** をセキュア化するよう設定され、承認の処理に独自のロールのリストを提供します。

5.5.2. 仕組み

この例では、**idp-domain** セキュリティードメインの **DatabaseLoginModule** によって使用されるデータベースに以下のユーザーが追加されています。

表5.6 idp-domain ユーザー

ユーザー名	パスワード	ロール
Eric	samplePass	all
Amar	samplePass	AB
Brian	samplePass	C
Alan	samplePass	

EAP-IDP の起動時、管理インターフェースが起動した後に **security** サブシステムが含まれるサブシステムとデプロイされたアプリケーションが起動します。**security** サブシステムには **idp-domain** と **IDP.war** が含まれます。**idp-domain** はデータベースに接続し、**DatabaseLoginModule** ログインモジュールに設定されたようにユーザー名、パスワード、およびロールをロードします。機密情報が **DatabaseLoginModule** ログインモジュールの設定にプレーンテキストで保存されないようにするため、データベースのパスワードなどの一部のフィールドを隠すようパスワードハッシュが設定されます。**IDP.war** は 認証と承認に **idp-domain** を使用します。また、認証されたユーザーに SAML トークンを発行し、認証の対象となるユーザーに対して簡単なログインフォームを提供するため、**IDP.war** は **jboss-web.xml**、**web.xml**、**picketlink.xml**、および **jboss-deployment-structure.xml** を使用して設定されます。

EAP1、**EAP2**、および **EAP3** の起動時、管理インターフェースが起動した後に **security** を含むサブシステムとデプロイされたアプリケーションが起動します。**security** サブシステムには各インスタンスの **sso-domain** が含まれ、それぞれ **sampleAppA.war**、**sampleAppB.war**、および **sampleAppC.war** が含まれます。**sso-domain** は **SAML2LoginModule** ログインモジュールを使用するよう設定されますが、アプリケーションに依存して適切な SAML トークンを提供します。よって、**sso-domain** を使用するすべてのアプリケーションは適切に IDP に接続して SAML トークンを取得する必要があります。この場合、各 **sampleAppA**、**sampleAppB**、および **sampleAppC** は **jboss-web.xml**、**web.xml**、**picketlink.xml**、および **jboss-deployment-structure.xml** を介して設定され、IDP のログインフォームを使用して **IDP.war** から SAML トークンを取得します。

また、各アプリケーションは独自の許可されるロールでも設定されます。

表5.7 アプリケーション/SP ロール

アプリケーション/SP	許可されるロール
sampleAppA	all、AB
sampleAppB	all、AB
sampleAppC	all、C

認証されていないユーザーが **sso-domain** によってセキュア化された URL (すべてのアプリケーションの **/secure/*)** にアクセスしようとすると、ユーザーは**IDP.war** のログインページにリダイレクトされます。その後、ユーザーはフォームを使用して認証され、ユーザーのアイデンティティーとロール情報が含まれる SAML トークンが発行されます。ユーザーは元の URL にリダイレクトされ、SAML トークンが SP であるアプリケーションに提示されます。アプリケーションは SAML トークンが有効であることを確認し、SAML トークンによって提供されたロールとアプリケーションで設定されたロールを基にしてユーザーを承認します。ユーザーに SAML トークンが発行された後、同じ IDP を使用して

SSO によってセキュア化された別のアプリケーションでは、ユーザーはそのトークンを使用して認証および承認されます。SAML トークンの期限が切れ、無効になると、ユーザーは IDP から新しい SAML トークンを取得する必要があります。

この例では、Eric が **EAP1/sampleAppA/secure/hello.html** にアクセスすると、ログインのために **EAP-IDP/IDP/login.html** にリダイレクトされます。正常にログインした後、Eric にユーザー情報とロール **all** が含まれる SAML トークンが発行され、**EAP1/sampleAppA/secure/hello.html** にリダイレクトされます。この SAML トークンが **sampleAppA** に提示され、トークンの確認後にロールを基にして承認が行われます。**sampleAppA** は、ロール **all** および **AB** の **/secure/*** へのアクセスを許可します。 Eric は **all** ロールを持っているため、**EAP1/sampleAppA/secure/hello.html** にアクセスできます。

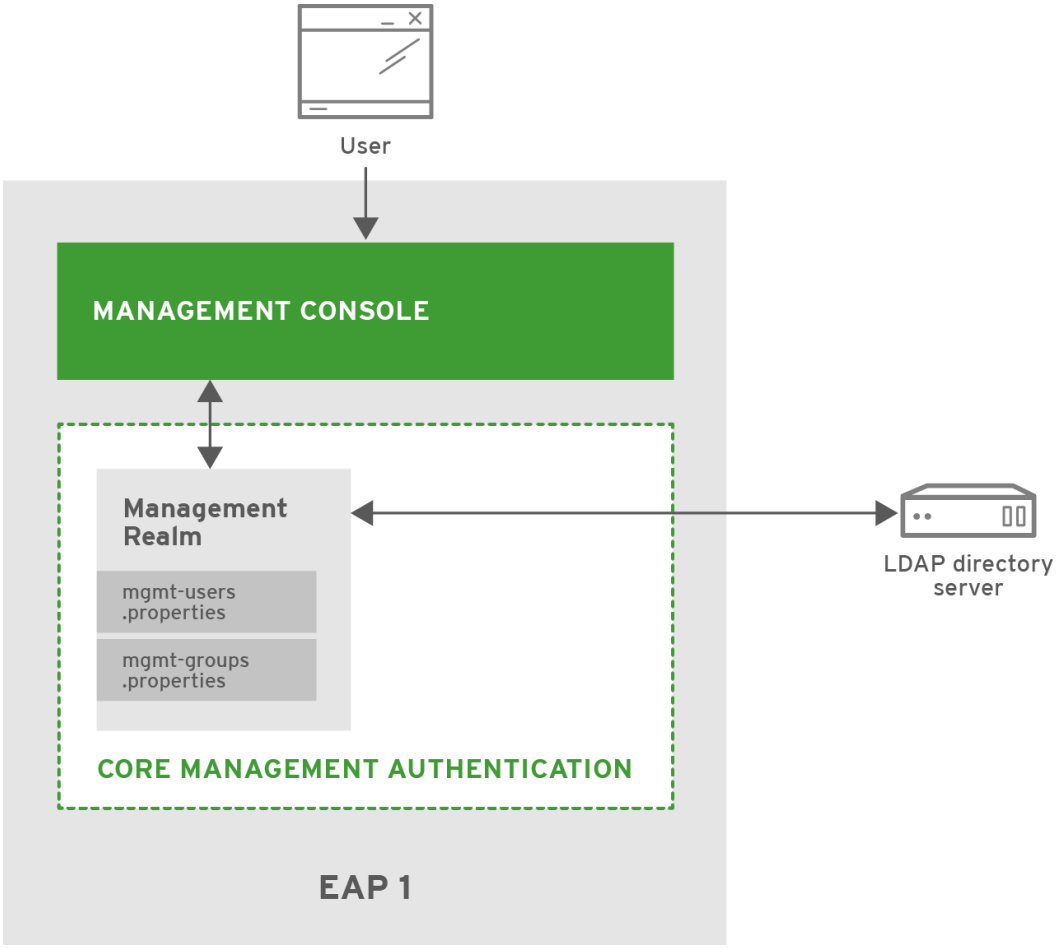
Eric が **EAP2/sampleAppB/secure/hello.html** にアクセスしようとすると、この SP に対しては認証されていないため、認証リクエストとともに **IDP.war** にリダイレクトされます。Eric はすでに IDP に対して認証されているため、その SP に対する新しい SAML トークンとともに **IDP.war** によって **EAP2/sampleAppB/secure/hello.html** にリダイレクトされ、再認証は必要ありません。**sampleAppB** によって新しい SAML トークンがチェックされ、ロールを基に承認が行われます。**sampleAppB** は、ロール **all** および **AB** の **/secure/*** へのアクセスを許可します。Eric は **all** ロールを持っているため、**EAP2/sampleAppB/secure/hello.html** にアクセスできます。Eric が **EAP3/sampleAppC/secure/hello.html** にアクセスする場合も同様に処理されます。

SAML トークンがグローバルログアウトによって無効になった後、Eric が **EAP1/sampleAppA/secure/hello.html** や、**EAP2/sampleAppB/secure/*** または **EAP3/sampleAppC/secure/*** 以下の URL に戻ろうとすると、再認証のために **EAP-IDP/IDP/login.html** へリダイレクトされ、新しい SAML トークンが発行されます。

Amar が **EAP1/sampleAppA/secure/hello.html** または **EAP2/sampleAppB/secure/hello.html** にアクセスしようとすると、Eric と同様に処理されます。Amar は **AB** ロールを持ち、**EAP1/sampleAppA/secure/*** および **EAP2/sampleAppB/secure/*** のみにアクセスできます。そのため、 **EAP3/sampleAppC/secure/hello.html** にアクセスしようとすると、SAML トークンの保持または取得が必要になりますが、**EAP3/sampleAppC/secure/hello.html** の閲覧は制限されます。Brian も同様ですが、**EAP3/sampleAppC/secure/*** のみにアクセスできます。Alan はロールを持たないため、サービスプロバイダーのセキュアな領域にアクセスしようとすると、SAML トークンの保持または取得が必要になりますが、閲覧はすべて制限されます。各 SP には、承認の有無に関わらず、すべてのユーザーが SAML トークンを取得しなくても閲覧可能なセキュアでない領域もあります。

5.6. LDAP と MANAGEMENTREALM の使用

ここでは、LDAP を使用して管理インターフェースをセキュアにする **ManagementRealm** を取り上げます。JBoss EAP インスタンス **EAP1** が作成され、スタンドアロンサーバーとして稼働しています。また、**EAP1** の **ManagementRealm** は LDAP を認証および承認に使用するよう更新されています。



5.6.1. セキュリティー

JBoss EAP は、LDAP および Kerberos を使用したセキュリティーレルムでの認証をサポートします。これには、既存の **ManagementRealm** が認証タイプとして **ldap** を使用するよう更新し、LDAP サーバーへのアウトバウンド接続を作成します。これにより、認証方法が **digest** から **BASIC / Plain** に変更され、ネットワーク上ではデフォルトでユーザー名とパスワードがクリアテキストで送信されます。

5.6.2. 仕組み

以下のユーザーが LDAP ディレクトリーに追加されています。

表5.8 管理ユーザー

ユーザー名	パスワード	ロール
Adam	samplePass	SuperUser
Sam	samplePass	

起動時、**EAP1** は **ManagementRealm** と管理インターフェースを含むコアサービスをロードします。**ManagementRealm** は LDAP ディレクトリーサーバーに接続し、必要に応じて管理インターフェースの認証を提供します。

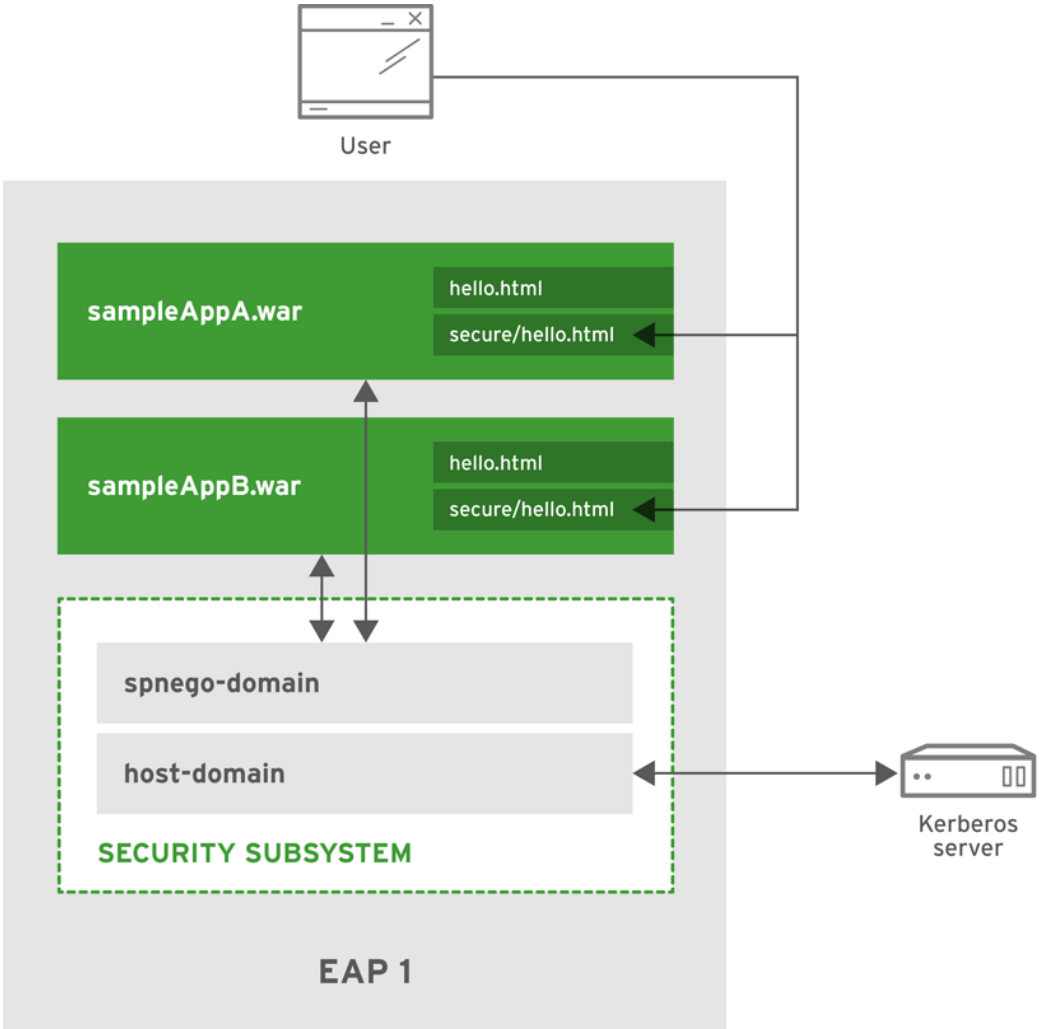
Adam が管理インターフェースにアクセスしようとすると、認証が強制的に実行されます。Adam のクレデンシャルは、認証に LDAP ディレクトリーサーバーを使用する **ManagementRealm** セキュリ

ティーレルムに渡されます。**EAP1** は認証後にデフォルトの簡単なアクセス制御を使用しているため、Adam のロールはチェックされず、アクセスが許可されます。Sam が管理インターフェースにアクセスしようとしても同様に処理されます。

Adam の認証後に RBAC が有効化されると、承認のために Adam のロールは管理インターフェースに戻されます。Adam は **SuperUser** ロールを持っているため、管理インターフェースへのアクセスが許可されます。サムが RBAC が有効になっている管理インターフェースにアクセスしようとすると、LDAP によって認証されますが、適切なロールを持っていないためアクセスは拒否されます。

5.7. KERBEROS を使う DESKTOP SSO を使用した WEB アプリケーションへの SSO の提供

ここでは、JBoss で Kerberos を使用して web アプリケーションに SSO を提供する方法について説明します。JBoss EAP インスタンス **EAP1** は作成済みで、スタンドアロンサーバーとして実行されています。**sampleAppA** と **sampleAppB** の 2 つのアプリケーションが **EAP1** にデプロイされています。両方のアプリケーションと **EAP1** は、Kerberos でデスクトップベースの SSO を使用して認証するよう設定されています。



5.7.1. セキュリティー

JBoss EAP は、SPNEGO と JBoss Negotiation による web アプリケーションでの SSO に対する Kerberos の使用をサポートします。Kerberos と SPNEGO に特化した情報は、「サードパーティーの SSO 実装」を参照してください。JBoss EAP とデプロイされた web アプリケーションが認証に Kerberos を使用するようにするには、2 つのセキュリティードメインを作成する必要があります。最初のセキュリティードメイン **host-domain** は、**kerberos** ログインモジュールで設定され、

Kerberos サーバーへ接続します。これにより、JBoss EAP のコンテナーレベルでの認証が可能になります。2 つ目のセキュリティードメイン **spnego-domain** は、2 つのログインモジュールで作成されます。その 1 つは **spnego** ログインモジュールを使用して **host-domain** に接続し、ユーザーを認証します。もう 1 つは別のログインモジュール (プロパティーファイルを使用してユーザーをロールにマップする **UsersRoles** など) を使用して、承認の決定に使用するロール情報をロードします。

これら 2 つのログインモジュールは、**password-stacking** を利用して、承認モジュールのユーザーとロールを認証モジュールのユーザーにマップします。**EAP1** は **host-domain** と **spnego-domain** の両方で設定されます。アプリケーションは、**spnego-domain** を使用して認証を実行し、承認のためにユーザーのロールを取得するよう、 **web.xml** および **jboss-web.xml** を介して設定されます。また、セクリティートークンをオペレーティングシステムからブラウザーに渡せない場合のために、FORM 認証をフォールバック認証としてアプリケーションに設定することもできます。FORM 認証がフォールバックとして設定された場合、追加のセキュリティードメインを追加してサポートする必要があります。このセキュリティードメインは Kerberos と SPNEGO には依存せず、FORM 認証のみ (ユーザー名とパスワードの検証およびロール) をサポートする必要があります。各アプリケーションは、**spnego-domain** を使用するよう設定され、FORM 認証をフォールバックとして提供するよう設定されます。また、各アプリケーションはパス **/secure/*** をセキュアにするよう設定され、承認の処理に独自のロールリストを提供します。

5.7.1.1. 仕組み

以下のユーザーが Kerberos サーバーに作成されています。

表5.9 Kerberos ユーザー

ユーザー名	パスワード
Brent	samplePass
Andy	samplePass
Bill	samplePass
Ron	samplePass

以下のロールは、 **password-stacking** オプションを **useFirstPass** に設定してチェーンされた追加のモジュールを介してユーザーにマップします。

表5.10 ユーザーロール

ユーザー名	ロール
Brent	all
Andy	A
Bill	B
Ron	

各アプリケーションには以下のロールが設定されています。

表5.11 アプリケーションロール

アプリケーション/SP	許可されるロール
sampleAppA	all、A
sampleAppB	all、B

起動時に、**EAP1** はコアサービスをロードしてから**security** およびその他のサブシステムをロードします。**host-domain** は Kerberos サーバーへの接続を確立し、**spnego-domain** は **host-domain** に接続します。**sampleAppA** と **sampleAppB** がデプロイされ、認証のために**spnego-domain** に接続します。

Brent は Kerberos でセキュア化されたコンピューターにログインしています。Brent はブラウザーを開き、**sampleAppA/secure/hello.html** へのアクセスを試みます。このページはセキュアであるため、認証が必要になります。**EAP1** は、Brent のコンピューターに設定されている Kerberos Key Distribution Center (Kerberos サーバー) へのキーを要求するリクエストを送信するよう、ブラウザーに指示します。ブラウザーがキーを取得した後、**sampleAppA** に送信されます。**sampleAppA** はチケットを **spnego-domain** に送信します。そこでチケットがアンパックされ、設定済みの Kerberos サーバーとともに **host-domain** によって認証が実行されます。チケットが認証されたら、Brent のロールは **sampleAppA** に戻され、承認が実行されます。Brent は **all** ロールを持っているため、**sampleAppA/secure/hello.html** にアクセスできます。Brent が **sampleAppB/secure/hello.html** にアクセスする場合も同じ処理が発生します。**all** ロールを持っているため、アクセスが許可されます。Andy と Bill にも同じ処理が発生しますが、Andy は **sampleAppA/secure/hello.html** のみにアクセスでき、**sampleAppB/secure/hello.html** にはアクセスできません。Bill はこの逆で、**sampleAppB/secure/hello.html** のみにアクセスでき、**sampleAppA/secure/hello.html** にはアクセスできません。Ron は **sampleAppA/secure/hello.html** または **sampleAppB/secure/hello.html** の認証に成功しますが、ロールを持たないためどちらにもアクセスできません。

オフィスのネットサークに接続する個人のラップトップなど、Andy が Kerberos でセキュア化されていないコンピューターから **sampleAppA/secure/hello.html** にアクセスしようとすると、フォールバックのログインとして FORM ログインページに転送されます。Andy のクレデンシャルはフォールバックセキュリティードメイン経由で認証され、承認の処理が続行されます。

Revised on 2018-08-29 12:46:31 EDT