Oracleホワイト・ペーパー
2014年10月

# Oracle ADFのセキュリティ：OWASPのセキュリティ脆弱性トップ10への対処

## 概要

会社のセキュリティをおびやかしかねない粗悪なコードを書こうと懸命になっているアプリケーション開発者はいません。Open Web Application Security Project（OWASP）は、セキュアなソフトウェアの記述方法を開発者に示すため、また開発者のセキュリティ意識を啓発するために、毎年特定されたWebアプリケーションの重大なセキュリティ脆弱性のトップ10リストを公開しています。

OWASPの脆弱性トップ10リストはテクノロジーに依存するものではなく、言語やフレームワークに固有の例、説明、ヒント、アドバイスなどは含まれません。本書では、2013年版OWASPトップ10に挙げられている上位10個のセキュリティ脆弱性のそれぞれに適用できる、Oracle Application Development Framework（ADF）のフレームワーク固有のヒントやアドバイスを提供します。

## 免責事項

また、Oracle ADFで使用できるセキュリティ・オプションおよびセキュリティ機能のうち、2013年版のOWASPのセキュリティ脆弱性トップ10リストで公開されるセキュリティ・リスクの軽減に役立つものについて説明します。なお、本書に記載した一連の推奨事項がすべてではありません。また、本書の提案項目をすべて実装しても、OWASPトップ10にリストされているすべてのセキュリティ上の脅威に対する十分な保護が得られる保証はありません。セキュアなアプリケーションを開発する責任を第三者や1つの文書に委ねることはできないため、ここに免責事項を掲載させていただきます。本書は、アプリケーションのセキュリティを実装するときに使用できるOracle ADFのセキュリティ識別ツールおよび機能についての知識を持つ開発者を支援することを目的としたものであり、正式なコード・レビュー・プロセスに取って代わるものではありません。

## はじめに

「The Open Web Application Security Project（OWASP）は、信頼できるアプリケーションの開発・購入・運用の推進を目的として設立されたオープンなコミュニティです」

- 2013年版OWASPトップ10[1]

Oracle ADFには、Java EE認証およびJava Authentication and Authorization Service（JAAS）を利用して認可を行うADF Securityという統合型アプリケーション・セキュリティ・フレームワークが含まれています。The ADF Security機能は、Oracle Platform Security Servicesベースのセキュリティをアプリケーションに組み込むための宣言的で視覚的な開発環境を提供します。Oracle ADF SecurityとOracle Platform Security Servicesを組み合わせて使用することで、開発者は保護する*方法*よりも保護すべき*対象*に注意を向けることができます。セキュリティは一方通行の道路ではありません。バインド・タスク・フローなど、Oracle ADFの多くの機能はもともとセキュリティ機能として設計されたものではありませんが、セキュリティ意識の高いアプリケーション開発者が既知のセキュリティ*脅威*からの保護を目的に使用することもあります。
Open Web Application Security Projectは、何に対して保護を行う必要があるかを開発者に示すために、年に1度、その年に特定されたもっとも重大な10個のセキュリティ脆弱性を文書にまとめて発行しています。

10個のセキュリティ脆弱性に対処したからといって総合的なセキュリティが実現するわけではありませんが、今あるセキュリティ*脅威*に対する意識を高めるきっかけにはなります。本書では、OWASPが2013年版としてまとめたセキュリティ脆弱性およびセキュリティ・リスクにOracle ADFで対処する方法について説明します。

## 2013年版OWASPのセキュリティ脆弱性トップ10

本書は、2013年版としてOWASPがまとめたセキュリティ脆弱性に対応しています[2]。さいわい、2013年版のOWASPトップ10リストは、順位が入れ替わっていることを除き、これまでに公開されたリストとほとんど変わりません。リストされているセキュリティ*脅威*はおそらくもっとも深刻な*脅威*であり、アプリケーション開発者はこれらの*脅威*を認識し、これらの*脅威*に対する保護を行う必要があります。

本書では、2013年版のOWASP脆弱性リストに記載されているセキュリティの*脅威*からADF Webアプリケーションを保護するための一助となるよう、Oracle ADFでの慣例と、次に示す*脅威*に対するリスクを軽減するための戦略について説明します。

1. **SQL インジェクション –** ユーザーがアプリケーションのユーザー・インタフェースに SQL コマンドを追加し、接続されているデータベースに対してそのコマンドを実行させる攻撃。アプリケーションに自由に SQL を入力できるようになっている場合は、機密データを公開、削除、操作するといった有害なコマンドをユーザーが誤って、または意図的に実行できるというリスクが存在します。
2. **認証とセッション管理の不備 –** 独自開発したアプリケーション機能でセッション・ハンドリングや認証を行っている場合は、トークンまたは資格証明が保護されていないと、アプリ

[1] https://www.owasp.org/images/7/79/OWASP_Top_10_2013_JPN.pdf
[2] https://www.owasp.org/images/7/79/OWASP_Top_10_2013_JPN.pdf

ケーション・ユーザーのなりすましや、セッションの乗っ取りが発生するリスクがあります。

3. **クロスサイト・スクリプティング（XSS）** – HTML スニペットや JavaScript をアプリケーションに入力し、他のユーザーがリクエストした Web ビューに表示させる攻撃。この種の攻撃の標的にされやすいのは、ユーザー同士で情報を共有できるようになっているディスカッション・フォーラムやソーシャル・ネットワーキング・サイトです。

4. **安全でないオブジェクト直接参照** – 印刷形式のレポートやインボイスを認証済みユーザーに提供するアプリケーションがありますが、ファイル参照が保護されていない場合やアドレス体系から予測できてしまう場合は、ユーザーが他のユーザーのドキュメントにアクセスできるというリスクが存在します。

5. **セキュリティの構成ミス** – いろいろな意味で、セキュリティは構成可能な機能です。ユーザーID は構成可能な ID ストアに保管され、パーミッションは構成可能なポリシー・ストアに保持されます。オプションは、デバッグを行う目的で開発者が一時的にセキュリティを無効化するために存在する場合があります。デフォルトでセキュアな状態にするというアプローチに従っていないアプリケーションの場合は、本番環境のセキュリティ構成にミスがあると、機密情報や機密データが未認可ユーザーに開示されてしまうリスクがあります。

6. **機密データの露出** – データは公開してよいものばかりとは限らないため、未認可ユーザーに機密情報を開示しないように注意する必要があります。セキュリティ構成を誤ったりセキュアでないデフォルトを選択したりすると、データ漏えいのリスクにつながる場合があります。

7. **機能レベルのアクセス制御の欠落** – "出荷すべきかしないべきか"という質問には、認可済みユーザー（オンライン・ショップまたは工場のユーザー）が答える必要があります。アプリケーション内では注文の出荷処理が関数にカプセル化されており、ユーザーの代わりにアプリケーションがこれを起動します。したがって、機密性の高い関数では、認証済みユーザーに対する認可を実施する必要があります。

8. **クロスサイト・リクエスト・フォージェリ（CSRF）** – 認証済みユーザー・セッションのコンテキスト内で HTTP リクエストを偽造しようとする攻撃。たとえば、表示されたリンクをユーザーがクリックすると、本人に代わって製品が発注されてしまうことがあります。CSRF はほとんどの場合、ロード時に自動的にリクエストを発行するユーザー・ページに攻撃者が HTML コンテンツを注入するクロスサイトスクリプティング（XSS）という手段を用いて実装されます。

9. **既知の脆弱性を持つコンポーネントの使用** – 比較的大規模なアプリケーションを開発者がまったくゼロから構築するというケースは極めてまれです。Java EE アプリケーションには、永続性エンジン、Web フレームワーク、JavaScript ライブラリ、Java ライブラリなど、なんらかのサード・パーティ製コンポーネントが使用されているのが普通です。こうしたコンポーネントでの欠陥はバグ・レポートとして公表されるか、修正済みバグ・リストとして新しいバージョンのドキュメントに記録されます。このようなフレームワークやライブラリの古いバージョンを使用しているアプリケーションは、こうした既知の脆弱性によってリスクにさらされます。

10. **未検証のリダイレクトとフォーワード** – Java EE アプリケーションは、モノリシック・アプリケーションとして構築することもできますが、保守性やパフォーマンスを向上させるために、個別の Java EE デプロイメントを使用することがよくあります。これらは、ユーザー・アプリケーションと認証コンテキストを渡しながら相互にコールし合います。信頼性があるとみなし、受信したリクエストの検証を行わない Java EE モジュールは、不正アクセスや有効なアプリケーション・コンテキストの外からアクセスされるリスクを招きます。

## セキュリティ意識の啓発と教育

お金で買える一番のアプリケーション・セキュリティは教育です。開発者やプロジェクト・リーダーは、セキュリティの問題に対する意識を高く持ち、セキュアなコーディング方法を理解している必要があります。トレーニングでは、潜在的なリスクについて詳しく説明するとともに、開発プラットフォームやデプロイメント・プラットフォームが搭載する、脆弱性攻撃の軽減に役立つ機能についても説明する必要があります。

アプリケーションのセキュリティの設計においてもっとも重要な原則は、セキュリティを仕様としてデフォルトで実装することです。セキュアなコーディングに関するガイドラインを用意し、使用するツールやプラットフォームに関係なく、すべての開発組織がこれを遵守および適用する必要があります。

デフォルトによるセキュリティのよい例としては、停電時のエレベータに対して誰もが予想する動作があります。それは、ブレーキを解除するのではなく、エレベータ室内にいる乗客の安全を確保するためにブレーキをかけるという動作です。しかし、この動作をデフォルトとして定義しておかなければ、ブレーキをかける必要があるかどうかエレベータにはわかりません。そのため、外部からの攻撃を阻止する方法について検討する前に、アプリケーションを内部から保護するために必要なセキュアなデフォルトとは何かを突き止める必要があります。ただし、これはトレーニングと啓発を行わなければうまく行きません。

## “独自構築”に伴うリスク

アプリケーションに必要なセキュリティ機能が、プラットフォーム提供またはフレームワーク組込みのセキュリティ・ツールセットからいつもすぐに見つかるとは限りません。その結果、開発プロジェクトでは“セキュリティが独自構築される”ことが珍しくありません。標準に準拠していない固有のセキュリティ・インフラストラクチャを使用する既存のシステムに取って代わるアプリケーションの場合は特にそうです。その例としては、実行時のユーザー・プロビジョニングおよびリソース権限付与に、データベース表に基づく認証および認可を組み合わせるというものがあります。

セキュリティを独自に構築する場合は、セキュリティ・レイヤーの品質保証、アプリケーション・セキュリティの伝播およびシングル・サインオンも独自対応になり、セキュリティ・レイヤーのバグ修正やメンテナンスが自己責任になるというリスクがあります。

開発者全員がセキュリティに詳しいとは限りませんが、カスタム・セキュリティ・レイヤーはセキュリティに詳しい人が構築する必要があります。

既存のセキュリティ・ソリューションの調査に時間を費やすことには、それだけの価値があるはずです。既存のソリューションをカスタム・アプリケーションに適用するほうが、エラーが発生しやすいメカニズムを自作するよりも簡単で費用対効果が高いこともあります。

## Oracle Platform Security Services（OPSS）

Java EEのセキュリティ・モデルは移植性が高くAPIの一貫性も優れていますが、このモデルではリクエストURIごとにリソースが保護され、パラメータやサーバー側リダイレクトは考慮されないため、最新のWebアプリケーションに対応する完全なセキュリティ・ソリューションとは言えません。認可機能はJava Authentication and Authorization Service（JAAS）のほうが優れていますが、Java EEとの統合性は高くありません。

JAASで認証を実行する場合、認証済みユーザーの管理にJava EEコンテナは必要ありません。

OPSSは、Java EEのセキュリティではカバーしきれない部分を補完するためにオラクルが提供しているベンダー固有のセキュリティ・プラットフォーム・サービスで、認証、認可、監査、ロールおよび資格証明の管理に対応した移植可能で一貫性のある標準ベースのソリューションを提供します。OPSSは、Oracle WebLogic Server、Oracle SOA、Oracle WebCenterおよびOracle Web Service Managerのセキュリティをオラクル Fusion Middleware内で統合します。

OPSSを使用すると、両者の優れた部分、すなわちコンテナ管理の認証による容易な認証とJAASがサポートする優れた認可機能をアプリケーション開発に取り入れることができます。Oracle ADF SecurityフレームワークはOracle ADF開発プラットフォームに不可欠なものですが、このフレームワークではOPSSを使用して、Java EEに基づくユーザー認証とJAASに基づく認可を実行します。

注：Oracle Platform Security Servicesプラットフォームがもたらすメリットは、Webアプリケーション開発にJAASが統合されることだけではありません。詳しくは、『Oracle Fusion Middlewareアプリケーション・セキュリティ・ガイド』[3]を参照してください。

## ADF Securityの概要

Oracle ADF Securityフレームワークには、開発者によるセキュアなADFアプリケーションの構築を容易にする機能がいくつも用意されています。そのほとんどはそのまま使用できる宣言的機能です。

ユーザーが、保護されているADFバインド・タスク・フローか保護されているトップレベルのADFにバインドされたJSFページにアクセスすると、必ずJAASのセキュリティ・チェックがADF Securityによって自動的に実行されます。認証はADF Securityではなく Java EEコンテナによって処理されます。ただし、ユーザーのログイン・アクションおよびログアウト・アクションが処理されるタイミングについては、アプリケーション開発者が全面的に制御できることに変わりはありません。

ユーザー・インタフェース・コンポーネントおよびタスク・フロー・アクティビティに対する認可をADF Securityで適用する方法については、後ほど説明します。Java EEコンテナ・セキュリティ上でOPSSを活用するOracle ADF Securityは、アプリケーション・ユーザー・パーミッションを安全に保管するために、ファイル・ベース、RDBMSベースおよびLDAPベースのポリシー・ストアをサポートしています。

重要な点は、ADF Securityは、セキュリティの実装という設計時の開発側面を実行時のセキュリティ管理やメンテナンスから切り離すことで、懸念と責任を分離できるように設計されているということです。

**注：**参考資料については後述しますが、2010年にMcGraw Hill社より刊行された『Oracle Fusion Developer Guide: Building Rich Internet Applications with Oracle ADF Business Components and ADF Faces』[4]という書籍の第21章に、ADF Securityについての詳しい説明があります。

[3] http://docs.oracle.com/cd/E48246_01/core.1111/b56235/toc.htm

[4] Oracle Fusion Developer Guide, McGraw Hill 2010, ISBN 9780071622547

## セキュリティ設計パターン

ソフトウェア工学における設計パターンとは、ベスト・プラクティスの完成予想図であり、効率的で堅牢かつスケーラブルなソフトウェアの記述方法を示すものです。設計パターンは実装に固有のものではなく、文書化されたベスト・プラクティスを、アプリケーションの構築に使用する任意の言語または任意のプラットフォームで実装できるよう、十分なガイドを提供するものです。次の項では、広く知られているパターンをいくつかピックアップして簡単に紹介します。

### パターン：多重防御

セキュリティは単一のレイヤーのみに実装すべきではなく、他のレイヤーにも実装する必要があります。たとえば、ユーザーのデータ入力の検証は、利便性と即答性の面からビュー・レイヤーで行うべきですが、ビジネス・サービス・レイヤーおよびデータ・レイヤー（データベースなど）でも実施します。Oracle ADFのアーキテクチャは複数のテクノロジー・レイヤーで構成されていますが、それぞれのレイヤーでADF Securityを使用して、ユーザー認可をJAASポリシーに対してチェックできるようになっています。

注：本書では主としてOWASPのセキュリティ脆弱性トップ10への対策について取り上げ、ADF Securityについては詳しく説明しません。ADF Securityについての個別の概要、およびハンズオン・チュートリアルは、“Security for Everyone”というOracle Magazineの記事を参照してください[5]。ADF Securityの概要を説明しているビデオは、ADF Insiderの“ADF Security”を参照してください[6]。

### パターン：最小権限アクセス

アプリケーション内で作業を行うユーザーに、基盤となるデータベース・システムの管理者権限を割り当てる必要はありません。その代わり、アプリケーション・ユーザーは、各自のタスクを終了させるために必要な最小限の権限でアプリケーションを実行する必要があります。

### パターン：単一アクセス・ポイント

バッキンガム宮殿の警備には何人の警備員が必要でしょうか。きっと、建物や敷地の入り口に警備担当として大勢の警備員が配置され、これを支える組織的なインフラストラクチャもあるでしょう。アプリケーションのエントリ・ポイントが増えるほど、防御すべき攻撃面は大きくなります。アプリケーションのエントリ・ポイントは1つであることが理想です。Oracle ADFでは単一アクセス・ポイントを簡単に実装できます。バインド・タスク・フローを使用してそのURL起動プロパティをfalseに設定するか、バインド・タスク・フローをページ・フラグメントとともに使用するだけです。

アプリケーションに複数のアクセス・ポイントが必要な場合はどうすればよいでしょうか。アーキテクチャ上の要件として複数のアクセス・ポイントを許可する必要がある場合（内部監査で使用するアプリケーションの一部を外部のユーザーも使用できるようにする場合など）は、コンパートメント化を検討する必要があります。

コンパートメント化を行うには、それぞれのアクセス・ポイントをモジュールとして扱い、認可のチェックを受けずにユーザーがモジュール間をナビゲートできないようにします。OWASPのセキュリティ脆弱性で10位にランクされている“未検証のリダイレクトとフォーワード”では、誤ったコンパートメント化を行った場合のリスクが指摘されています。

[5] http://www.oracle.com/technetwork/issue-archive/2012/12-jan/o12adf-1364748.html

[6] http://download.oracle.com/otn_hosted_doc/jdeveloper/11gdemos/AdfSecurity/AdfSecurity.html

### パターン：チェックポイント

ベルリンにあるチェックポイント・チャーリーは、スパイや捕虜の交換に使用される東と西の接点として冷戦時代に有名になりました。チェックポイントの役割は、認可済みの通信がレイヤー間（Webアプリケーションのビュー・レイヤーとビジネス・サービス・レイヤーとの間など）で行われるようにすることです。これは、ユーザーがリクエストしたアクションをそのユーザーが処理できるかどうかを判断するセキュリティ・ポリシー・コールでメソッド・コールをラッピングすることで実現できます。Oracle ADFでは、ADF Securityポリシーへのダイレクト・アクセスを使用して、あるいはビュー・コンポーネントまたはコントローラ・コンポーネントで式言語（EL）を使用して宣言的に、チェックポイントをJavaで作成できます。

### パターン：完全表示と限定表示

ユーザーが異なれば責務も異なるため、ユーザーがアプリケーション内で表示できる内容はそれぞれの責務を正確に反映したものである必要があります。情報のフィルタ処理はUIのみで使用するべきではなく、通知メッセージやエラー・メッセージでも使用する必要があります。たとえば、データベース制約の詳細情報を含むエラー・メッセージは一般のエンドユーザーの役には立ちませんが、顧客サポートを補助する情報として持っておくと極めて便利です。ただし、詳細なエラー・メッセージやスタック・トレースにはソフトウェアや実装に関する有益な情報が含めることもできるため、システムの脆弱性を攻撃しようと企んでいる人を喜ばせることにもなります。

### パターン：監査

ユーザーは、管理者がセキュリティ違反についてシステム監査を行えるようにするために、または法的規制を遵守するために、自分のアクションを記録に残す必要があります。たとえば、診断のために後から使用できるログが何もなかったら、SQLインジェクションが試行されたかどうかをどのようにして判断できるでしょうか。データ変更についても同じことを行う必要があるでしょう。表に対する最新の変更のみを追跡しているとしたら、データがどのように変化していったかを理解できません。構築するアプリケーションによって異なりますが、ADF Business Components（ADF BC）のエンティティのdoDML、またはデータベースのトリガーのようなフック・ポイントを見つける必要があります。

### パターン：ロール

セキュリティ管理には人による操作と見落としが伴うため、アプリケーション・セキュリティ・チェーンの弱点になることがあります。複雑さはミスを招くため、簡素化することが大切です。ロール・ベースのアクセス制御を使用するほうが、個別にアクセス権を付与する方法でシステムを管理するより簡単です。Oracle ADFアプリケーションでは、アプリケーションの一部として定義するロールにセキュリティを付与します。このようなアプリケーション固有のロールはエンタープライズ・ロールにマッピングされ、アプリケーションの範囲外のユーザーにはこのエンタープライズ・ロールが割り当てられます。このようにして、アプリケーションのセキュリティ管理はID管理インフラストラクチャから切り離されるため、実際にタスクを実行するのが誰になるかを考える必要がなくなり、アプリケーションの特定部分の実行に必要な権限は何かということだけを考えながら開発を進めることができます。エンタープライズ・ユーザーという概念は、特に比較的大規模な組織において、管理者が従業員のロールを理解する必要性や従業員の入社や退社の時期に配慮する必要性をなくすうえで役立ちます。

ロール・ベースのセキュリティを使用すると、簡素さと安全性が確保されます。Oracle ADFではロール・ベースの認可がサポートされるだけでなく、ロールのネストとエンタイトルメントもサポート

されます。エンタイトルメントとは、1つのgrant文で発行できるように複数のパーミッションをグループにしたものです。

### パターン：セッション

セッションと言ったときにJava EEの開発者が思い浮かべるのは通常、ビュー・レイヤーのHTTPセッションです。ただし、セッションも認証済みユーザーに対してセキュリティ・コンテキストを提供し、それがアプリケーションの複数レイヤーにまたがることがあります。

Oracle ADF Securityでは、Java EEのコンテナ管理の認証を使用してユーザーとユーザーのグループ・メンバーシップを追跡します。ユーザーの認可は、セッション・コンテキストに基づいて自動的に実施されます。

## ADF Securityのレイヤー

ADFには、保護を追加できる場所がいくつかあります。図1に示しているのは、Oracle ADFアプリケーションを構築するときに通常使用される保護領域の例です。

| ビュー/コントローラ | ADF Security | ビジネス・サービス | データベース |
|---|---|---|---|
| •認証<br>•ページ認可<br>•フィールド認可<br>•入力検証 | •ページ・セキュリティ<br>•タスク・フロー・セキュリティ | •ビジネス・メソッド認可<br>•CRUD認可<br>•入力検証 | •DML認可<br>•読み取り認可<br>•PLSQL認可 |

図1：ADF Securityのレイヤー

**ビュー/コントローラ** – ビュー・レイヤーはOracle ADFアプリケーションのユーザー・インタフェースに相当します。ADFにバインドされているビューは、トップレベルのドキュメントであれば直接アクセス制御の対象となり、ADFリージョンで公開されるフラグメントであればADF Controllerを通じた間接認可管理の対象となります。いずれにしても、認証されていないユーザーについては、保護しているビューが初めてアクセスされるときに、ビュー・レイヤーの認証チェックが自動的にトリガーされます。また、ユーザーが表示したりアクションを起動したりできないコンポーネントを非表示にする場合にも、ビュー・レイヤーを使用できます。コンポーネントの無効化、非表示、またはスクリーンからの削除を選択できる場合は、後の方のオプションを選択する必要があります。ビュー・レイヤーのコンポーネントは、悪意のあるデータ入力がビジネス・サービスに渡されないようにするための入力検証を実行するときに使用できます。

**ADF Security** – ADF SecurityはADFアプリケーションの開発者がビュー、コントローラ・アクション、ADF BCビジネス・サービスに対して認可を適用するときに使用するツールです。ADF Securityはビュー、タスク・フローおよびビジネス・オブジェクトのアクセス制御を定義するための宣言的なセキュリティ・フレームワークですが、JAASと連携して認可を行うプロセスを簡素化するAPIも提供します。Oracle ADF SecurityおよびOracle Platform Security Servicesを理解するうえで不可欠なのは、*ユーザーID*、*エンタープライズ・ロール*、*アプリケーション・ロール*という3つの主要な概念です。

**ユーザーID**：エンタープライズ内のユーザーを定義します。ユーザー（会社の従業員など）は通常、組織内のアプリケーションでの認証に使用するユーザー名/パスワードのペアを1つ所有します。ユーザーIDにはユーザーが誰であるかのみを定義し、アクセス権限は一切定義しません。

システムのデプロイメントや管理を容易にするために、管理者は通常、ユーザーを**エンタープライズ・ロール**にまとめます。これを使用すると、エンタープライズ・リソースにアクセスするときの要件が類似している複数のユーザーをまとめて管理できます。たとえば、Employeesという名前のエンタープライズ・ロールに従業員をグループ化し、企業内の全従業員向けセルフサービス・アプリケーションへのアクセス権を付与することができます。管理の面から見ると、エンタープライズ・ロールに対してユーザーの追加または削除を行うほうが、個々のユーザーのアプリケーション権限を維持管理するよりも簡単です。

**アプリケーション・ロール：**アプリケーション固有のもので、エンタープライズ・ロールに定義されているユーザーに権限を付与するときに使用します。アプリケーション・ロールを使用すると、同じエンタープライズ・ロール（Employeesなど）に属するすべてのユーザーに、さまざまなアプリケーションに定義されている固有のアクセス権限を付与することができるようになります。エンタープライズ・ロールに属するユーザーがアプリケーション内で作業する場合は、エンタープライズ・ロールにアプリケーション・ロールを付与する必要があります。ユーザーに直接アプリケーション・ロールを付与することもできますが、そうすることはまれであり、適切なプログラム設計とは見なされません。

**ビジネス・サービス –** ADFには、アプリケーションで必要となるビジネス・データやビジネス・サービスにアクセスするためのオプションがいくつかあります。もっとも使用されるビジネス・サービス・オプションは、ADF Business Componentsのオブジェクト・リレーショナル・マッピング・レイヤーです。ビジネス・エンティティを保護し、権限の変更と読取りを制御するために、ADFのこの部分はADF Securityと緊密に統合されています。ADF SecurityにはプログラミングAPIも用意されているため、プログラマは、ADF Business Componentsフレームワーク内でJavaコードを使用してより具体的なセキュリティ・ルールを開発することもできます。たとえば、新規エンティティの場合、ルールで特定のエンティティ属性の更新を許可しても、レコードが保存された後は属性の更新されないようにすることができます[7]。同様に、ADF BCのサービス・レイヤー内でライフサイクルまたはカスタム・メソッドを起動する前にADF Securityのパーミッションがチェックされるように定義することができます。

**データベース –** データベース（ここでは、具体的にSQLベースのRDBMSを指すのではなく、一般的な意味での永続ストアを指します）には、ADFアプリケーションで使用できるセキュリティ機能が用意されている場合があります。たとえば、Oracle Databaseにはラベル・セキュリティ、プロキシ・ユーザーのサポート、PL/SQLのAPIがあり、ADFと併用してデータを保護することもできます。

### 創造性

Webアプリケーションのセキュリティを確保する必要があるということと、開発者に創造性を発揮してもらうことは、何ら矛盾することではありません。どのようなアプリケーション開発テクノロジーを使用するにしても、枠にとらわれない発想力を持った開発者が、もともとセキュリティ用に設計された訳ではないフレームワーク機能を総合的なアプリケーション・セキュリティ戦略に組み込むといったことができると、Oracle ADFのアプリケーション・セキュリティを最大限に活用することができます。そのようなADFの機能は、次のとおりです。

[7] http://www.oracle.com/technetwork/developer-tools/adf/learnmore/76-insert-update-entity-protection-334421.pdf

**サーブレット・フィルタ** – ADFではWebユーザー・インタフェースにJavaServer Facesを使用し、標準のWebサーブレットを通じてJSFアクセスを行います。サーブレット・フィルタを使用すると、受信したリクエストをチェックしてリクエストをブロックまたはリダイレクトしたり、リクエストのパススルーを許可したりできます。

**JSFまたはADFのフェーズ・リスナー** – リクエスト・ライフサイクル内ではわずかに遅れますが、サーブレット・フィルタと同様に、JSFのフェーズ・リスナーを使用して受信リクエストをリスニングし、JavaServer Facesのリクエスト・ライフサイクルのコンテキスト内で応答することができます。たとえば、ビューに対する認可の実施やレンダリングする一連のUIコンポーネントの制御を、フェーズ・リスナーで実行することもできます。

**コンポーネント・バリデータ** - JSFでは、ユーザー入力コンポーネントに検証コンポーネントを宣言的に追加することができます。このようなバリデータは、ユーザーが入力した内容にスクリプト注入攻撃やSQLインジェクション攻撃などの怪しいパターンがないかどうかをチェックするときに使用できます。

**バインド変数** – SQL問合せでは、ユーザーが利用できるデータをフィルタするwhere句を、バインド変数を使用して構成できます。バインド変数を使用することがSQLインジェクション攻撃に対する保護になるというのは、幸運な副次効果です。

**WEB-INF** – WebアプリケーションプロジェクトのHTMLルート・フォルダの下にあるWEB-INFディレクトリに保管されているファイルはどれも、ブラウザのURLフィールドから直接アクセスできません。これらにアクセスするには、代わりにサーブレットを使用する必要があります。そのためこれは、開発者にこれらのリソースに対するアクセス制御機能を提供します。

**MDSカスタマイズ・クラス** – Meta Data Servicesは、Oracle ADFで使用されるカスタマイズとパーソナライズのエンジンです。カスタマイズ・クラスはJavaベースの決定ポイントです。特定のリクエストのためにビュー・オブジェクトまたはビジネス・オブジェクトにメタデータを追加すべきか変更すべきかは、ここで判定されます。カスタマイズ・クラスにはADFセキュリティ・コンテキストへのアクセス権が設定されているため、ユーザーに付与されているロールまたはパーミッションを使用して、特定の情報を追加するか非表示にするかを判定できます。

**タスク・フロー指向のアーキテクチャ** – ADFアプリケーションの構築とは、要するにタスク指向の開発です。ADF Securityは別にして、バインド・タスク・フローはセキュリティ強化に使用できる追加機能を提供します。まず、バインド・タスク・フローではアクセス・ポイントが必ず1つになるため、ユーザーがURLを操作して手当たり次第にコンテンツにアクセスするのを防げます。次に、タスク・フローを単独で使用できるのか、コンシューミング・アプリケーションで制御されるコンテキスト内のサーフェスでなければならないのかを、開発者が制御することができます。

**UIコンポーネントのrenderedプロパティ** – renderedプロパティはどのJSFコンポーネントでも公開されるものです。このプロパティはADFセキュリティ・コンテキストから取得される情報を使用してプログラム的に、または宣言的に設定できます。そのため、ページのユーザーが適切なロールまたはパーミッションを持っていない場合は、1つのコンポーネントまたはコンポーネントのグループをこのプロパティを使用してページから削除できます。

**注：**このあとは、OWASPトップ10に挙げられている脅威にセキュリティ意識の高い開発者がOracle ADFで対処するにはどうすればよいかについて説明します。トップ10がある場合はたいていトップ100もあるということを覚えておいてください。Oracle ADFでOWASPトップ10に対処することは、ADF Webアプリケーションのセキュリティを大きく前進させることではありますが、このチェックリスト1つで脆弱性がないと保証されるわけではありません。

## Oracle ADFによるOWASPトップ10への対処方法

ここからは、OWASPのセキュリティ脆弱性トップ10への対処方法について、2013年に公開されたOWASPの文書に挙げられている順に説明します。

### OWASP #1 - SQLインジェクション

サーバー側の問合せを定義または変更するデータがシステムに入力された場合は、SQLインジェクションのリスクを回避するためにすべてのデータを検証する必要があります。この要件に対して*アプリケーション*ではなく*システム*という言葉を使用している点に注意してください。最近は、アプリケーションという用語では、ユーザー操作の始まりと終わりを表す境界を十分に説明できなくなっています。アプリケーション・データの問合せや書込みには、さまざまなアクセス・チャネルを自由に使用できます。たとえば、Webサービスがモバイル・フロントエンドから使用されることがありますが、ダイレクトSQLやPL/SQL文はOracle Application Express（APEX）から使用され、SOAではデータベース・アダプタを使用してビジネス・プロセスの一環としてデータベースにアクセスする場合があります。これらのチャネルすべてを、SQLインジェクション攻撃から保護する必要があります。ADFを使用する場合は、SQL問合せの操作からADF Business Componentsモデルを保護することで、この保護を実現することができます。

#### ユーザー入力の検証

SQLインジェクションからの保護に最適な場所は問合せの定義を行うところですが、ADFの場合はたいていADF Business Componentsのビジネス・サービス・レイヤーです。Oracle ADFで使用されるビジネス・サービス・レイヤーをアプリケーション開発時に制御していない場合（サービスを使用している場合など）は、Webサービス・モデルによってリスクが緩和されることを期待し、Webサービス・コールに直接渡される入力フィールドにクライアント側検証を追加してリスク緩和を手助けする以外にできることはありません。

Oracle ADFを使用する場合は、ユーザー入力の検証を実行できるレイヤーが複数あります。その大半については、OWASP #3 – クロスサイト・スクリプティング（XSS）の項で詳しく説明しています。

通常、SQLインジェクション攻撃が試行されたかどうかは、ユーザーが入力した内容と、特定のコンテキスト内で許可されている既知のSQL式のリストとの比較によって検出されます。許可されている値やキーワードと比較してユーザー入力を検証する手法は、ホワイトリスト方式と呼ばれます。ホワイトリスト方式と対照的なのがブラックリスト方式です。ブラックリスト方式では、特定のコンテキストで許可されていない値やキーのリストと比較してユーザー入力をチェックします。

なお、通常はホワイトリスト方式が優先されます。なぜなら、この検証を通過しなかった場合はセキュリティ・ホールが開かれず、たいていは、有効であるにもかかわらずリストから漏れているSQLキーワードを入力したアプリケーション・ユーザーが困るだけだからです。

ユーザー入力とホワイトリストまたはブラックリストとの比較には、正規表現による検証を使用します。検証は、ADF Facesコンポーネントおよびバインディング・レイヤーに追加できます。

**ヒント：**すべてのユーザー入力フィールドについてSQLインジェクションがないかどうかを検証するのは合理的ではありません。データベースに永続化されるだけで実行されないデータ入力であれば、たとえそこにSQLキーワードが含まれていたとしても被害は発生しません。SQLインジェクションからの保護が必要な領域を特定して保護対策を実施することが、セキュリティ意識の高い開発者としての責任です。

**バインド変数の使用**

基盤となるデータベースへの問合せに使用されるSQL式は、ADF Business Componentsのビュー・オブジェクトに保持されます。通常SQLで行うように、ADF BCの問合せのフィルタ処理にはwhere句を使用します。

たとえば、ビュー・オブジェクトに定義された次の問合せを見てみましょう。

*Select EMPLOYEES.LAST_ID , EMPLOYEES.LAST_NAME from EMPLOYEES;*

ユーザー・インタフェースの入力コンポーネントにユーザーが直接入力した未検証の内容に基づいて実行時にwhere句を追加すると、問合せは下のようになります。

*Select EMPLOYEES.LAST_ID , EMPLOYEES.LAST_NAME from EMPLOYEES WHERE EMPLOYEES.DEPARTMENT_ID = 60;*

この場合は何も問題ありませんが、下のように定義される可能性もあります。この場合、最初は問合せだったものが最後は表全体の削除になってしまいます。

*Select EMPLOYEES.LAST_ID, EMPLOYEES.LAST_NAME from EMPLOYEES WHERE EMPLOYEES.DEPARTMENT_ID = 999 or 1=1; Drop table EMPLOYEES;*

もう1つは、被害は少ないながらも有害であることに間違いないSQLインジェクションの用法で、アプリケーションには表示されなくてもデータ・スキーマには存在するデータに関する情報を探り出そうとする例です。

たとえば、Steven King（話を簡単にするために、"King"という名前の従業員はシステムに1人しかいないものとします）の給与に関する情報を取得するために、下に示すようなSQL文字列を繰り返し使用することができます。

*Select EMPLOYEES.LAST_ID , EMPLOYEES.LAST_NAME from EMPLOYEES WHERE DEPARTMENT_ID=60 and exists (select "x" from SALARIES where name="King" and salary between 1000 and 20000);*

上の問合せから戻された結果がアプリケーション・ユーザー・インタフェースに表示されると、Steven Kingの給与は1000から20000の間であることが攻撃者に知られます。その後攻撃者は、必要な情報が得られるまで給与の範囲を狭めるでしょう。

エンティティの変更時や関数の実行時にどれほど適切に認可を適用しても、UIの入力フィールドを介したアプリケーションへのSQLインジェクションが成功するようでは何の役にも立たないことを、この例は示しています。

上に示したようなSQLインジェクション攻撃を回避するには、ユーザーがwhere句に入力したデータを検証します。さらによいのは、代わりにバインド変数を使用する方法です。

*Select EMPLOYEES.LAST_ID , EMPLOYEES.LAST_NAME from EMPLOYEES WHERE EMPLOYEES.DEPARTMENT_ID* = :DepartmentIdVar;

**動的なビュー・オブジェクトに注意**

前述したように、ADF Business Componentsの宣言的なビュー・オブジェクト開発手法とバインド変数による問合せのフィルタを組み合わせるのが、SQLインジェクション攻撃への保護対策として最善の選択肢です。しかしながら、実行時にビュー・オブジェクト定義を動的に作成することが必要となるビジネス・ケースは存在します。その場合、ビュー・オブジェクトによって実行されるSQL問合せのほとんどは、ユー

ザー入力を元に作成されます。このようなコードについては、特別な検査を実施し、合成された問合せにSQLインジェクション攻撃のフラグメントが含まれていないことを確認する必要があります。

ビュー・オブジェクトを動的に構築する際のSQLインジェクションのリスクを軽減するために、次のことを推奨します。

- SQL問合せ、WHERE句、ORDER BY文字列は別々に処理します。問合せ文が1つの文字列として入力されたら、WHERE、ORDER BY、UNION、GROUP BYなどのSQLキーワードで文字列を分割します。UNION文およびEXISTS文に含まれるSelect文字列は別々にチェックする必要があります。
- Javaの正規表現を使用して、構成された問合せ文字列と問合せの構成要素として許可しないSQL式のリストを比較して分析します。ユーザー入力として禁止される可能性がある式の例は“where”、“:=”、“==”、“;”（セミコロン）、“drop”、“grant”などです。逆に、許可されている値（ホワイトリスト）に対してチェックすることもできます。
- フィルタ値を追加するWHERE句ではバインド変数を使用します。引数値の入力にバインド変数を使用すると、SQLを問合せの引数として注入することができなくなります（問合せフィルタが異なる同じ問合せを再利用する場合）。WHERE句の文字列については、ブラックリストに挙げた‘;（セミコロン）’、“drop table”などの式が含まれていないかどうかを、正規表現を使用してチェックする必要があります。

### OWASP #2 - 認証とセッション管理の不備

ユーザー・セッションは、インスタンス固有のアプリケーション・データをユーザーのために保持するワーキング・コンテキストです。ユーザー・セッションが認証されている場合は、認可を実施し、ユーザーのパーミッションおよび権限の境界内でアプリケーションの機能を実行できるようにします。ユーザー・セッションの認証や管理はADF自体では処理されず、Web logic ServerなどのJava EEサーバー、すなわち専門家に委任されます。

それでも、ユーザー・セッションが正当な所有者に確実に所有されるように、デプロイ済みのADFアプリケーションについて検討すべきことがあります。

**パスワード –** パスワードは必ず暗号化形式で保護してIDストアに保存します。パスワードの質に対するポリシーを適用し、一定の長さがあり文字種および大/小文字が混在する強固なパスワードの設定を求める必要があります。また、パスワードのライフサイクルとして、ユーザーに頻繁なパスワード変更を求めたり、同じパスワードの連続使用を禁止したりする必要があります。パスワードの変更には1つのメカニズムを使用する必要があります。アプリケーションでパスワードを保存する場合は、メモリまたはデータベースのいずれに保存するとしても、暗号化されていない形式で保存しないようにする必要があります。

**SSL –** 認証は、ユーザーがログイン・フォームにユーザー名とパスワードを入力する場所で実行されます。このログイン・フォームではTransport Layer Security（SSL）を適用してユーザー資格証明を送信し、パスワードがクリア・テキスト形式で送信されないようにする必要があります。アプリケーションを所有していても、ネットワーク全体をエンド・ツー・エンドで制御できるとは限らないことに留意しをてください。ユーザーのブラウザとアプリケーション・サーバーの間のトラフィックはすべて識別可能だということを常に前提としましょう。

**認証 –** 認証の実行方法は、Java EEのBasic認証のような緩いものから証明書やシングル・サインオンを使用するより厳密なアプローチまで複数あります。保護を最大化するために、ADFアプリケーションでは厳密認証を使用するようにしてください。

**注：** Basic認証を切断することはできません。Basic認証はブラウザSSOとも呼ばれますが、それは、特定のレルムやドメインに対するクライアント認証がブラウザで行われるためです。アプリケーションのセッションを無効にした後でも、新たにリクエストを発行すれば新規の認証済みセッションが作成されるため、クロスサイト・スクリプティング攻撃やリクエスト・フォージェリ攻撃を仕掛けられる余地が大量に残ります。Basic認証から切断する方法は、ブラウザを閉じる以外にありません。

**信頼関係 –** Oracle ADFは、アプリケーションの規模やビジネス要件に応じてさまざまなアプリケーション・アーキテクチャをサポートします。"ピラー"とは、大規模なアプリケーションを複数の独立したADFアプリケーションに分割し、その1つ1つをJava EEアプリケーションとしてデプロイしたアーキテクチャです。分割するときに重要なのは、結合部でセキュリティを分断しないことです。ユーザー認証はJava EEアプリケーションごとに実施し、コール元とコール先のアプリケーション間に存在するトークンを通じて作成された信頼関係は利用しないようにする必要があります。このようなアーキテクチャでは、シングル・サインオンを使用する必要があります。

ID管理は1つのアプリケーションの範囲を超えて実施されるのが一般的であるため、アプリケーションの機能に含めるのではなく、管理レベルで実施する必要があります。独自のユーザー表やポリシー表を管理しているアプリケーションは、認証やセッション処理を実装する開発者がセキュリティに精通していない場合や倫理的ハッカーによる実装のテストを行っていない場合は脆弱である可能性があります。認証およびセッション管理は、専門のセキュリティ環境やサーバー環境に任せるのが常に最善の方法です。

**注：**シングル・サインオンはADF自体では処理されず、セキュリティ・プロバイダ（Oracle Web logic ServerのOPSS）に委譲されます。

**エラー・メッセージの適正化 –** 入力されたユーザー名とパスワードの組合せに基づくユーザー認証と、アカウント・ロック・フラグなどのチェックを追加で行うアプリケーション・ログインの場合は、認証が失敗した理由をユーザーに通知することがあります。エラー・レポートの内容が具体的すぎると、ターゲット・システムに関する有益な情報がハッカーの手に渡る恐れがあります。たとえば、入力されたパスワードが正しいかどうかを検証せずに、入力されたユーザー名に基づいてアカウントがロックされているかどうかを先にチェックするログイン・ロジックの場合、アカウントのステータスがロック済みになっていることを報告するエラーを表示すると、ハッカーは手元にあるユーザー名が有効であることを確認でき（これを元に他の有効なユーザー名を推定することができます）、ロックをかけるポリシーをシステムで使用していることがわかります。このような状況では、リクエストを認証できなかったことのみを伝えるのが、メッセージ形式としてはもっともセキュアです。認証に失敗したユーザーからサポート・チケットやリクエストを提出されるようなことがあっても、アプリケーション・ログを調査すれば詳しい情報を取得できます。

ADFでは、DataBinding.cpxファイルでカスタム・エラー・ハンドラを構成し、例外およびメッセージを独自にフィルタすることができます。カスタム・エラー・ハンドラは、再利用可能なバインド・タスク・フロー・プロジェクト内のDataBinding.cpxファイルではなく、デプロイするアプリケーションのDataBinding.cpxファイルで構成する必要があります。カスタム・エラー・ハンドラの作成方法および構成方法の例は、製品ドキュメントを参照してください。

### OWASP #3 - クロスサイト・スクリプティング（XSS）

前述したとおり、システムへのデータ入力は、データベースに永続化される前に検証する必要があります。特に、後でアプリケーション・ユーザー・インタフェースに再表示されるデータ入力は、クロスサイト・スクリプティングのリスクに曝されます。

有効なデータ入力が行われるようにするには、Oracle ADFでは次の戦略を実装する必要があります。

1. フリーテキスト入力フィールドの数を制限し、選択方式に置き換える
2. 2 つ以上のレイヤーですべてのユーザー入力を検証する
3. 出力データをエスケープする

## すべてのユーザー入力の検証

クロスサイト・スクリプティングを阻止するうえで重要なことは、データ入力の検証です。Oracle ADFでは、ブラウザ・クライアント上、ADF Facesコンポーネント上、ADFバインディング・レイヤー内、ADF Business Componentsなどのビジネス・サービス、およびデータベースで、JavaScriptを使用した入力検証を処理できます。

ビジネス・サービス・レイヤーは門番であり、データのアクセス方法に関係なく、すべてのデータ入力を検証する必要があります。他のアプリケーション・レイヤーに補完的なサポートを追加し、多重防御を実装したり、無効なデータ入力に関するタイムリーなフィードバックをエンドユーザーに提供したりできます。

### アプリケーション・セキュリティにおけるJavaScript検証

JavaScriptsはブラウザで実行されますが、アプリケーション開発者が制御できないブラウザはもともとJavaScriptsを実行する環境としては適していません。SQLインジェクション攻撃やクロスサイト・スクリプティング攻撃をJavaScriptで阻止することはできません。

たとえば、テキスト・フィールドに数値データ以外は追加入力できないようにして、SQLインジェクション攻撃とクロスサイト・スクリプティング攻撃から保護してみましょう。ADF Facesでは次のJavaScriptを使用することができます。

```
function onlyNumbers(evt) {
    var keyCode = evt.getKeyCode();
    isNumeric = (keyCode >= 48 && keyCode <= 57)
                ||(keyCode >= 96 && keyCode <= 105);
    var filterField = evt.getCurrentTarget();
    var oldValue = filterField.getValue();

    if (!isNumeric) {
        filterField.setValue(oldValue);
        evt.cancel();
    }
}
```

入力フィールドがフォーカスされているときにユーザーがコンピュータのキーボードのキーを押したら必ずJavaScript関数が起動されるようにするには、下に示すように、ADF Facesクライアント・リスナーの動作タグを入力コンポーネントに追加する必要があります。

```
<af:inputText label="Filter Character Keys" id="it1">
   <af:clientListener method="onlyNumbers" type="keyDown"/>
</af:inputText>
```

上のサンプルでは数値以外のデータ入力が非表示になっていませんが、このレベルのセキュリティで十分でしょうか。もちろん、十分ではありません。このサンプルには脆弱性の問題が少なくとも3つあります。

- JavaScriptを実行したときの動作はブラウザによって異なる可能性があり、あるブラウザで想定どおりに動作したスクリプトが別のブラウザで失敗することもあります。
- 知識の豊富なハッカーがページでのリスナーのレンダリングを無効にし、チェックを迂回することがあります。
- クライアントとサーバー間の通信がデバッグ・ツールによって傍受され、悪意のあるユーザーがSQLインジェクション・スクリプトまたはクロスサイト・スクリプティング・スクリプトを追加することがあります。

**では、ADF Faces（というより、ブラウザにデプロイされるすべてのアプリケーション）でのJavaScript検証は何に適しているのでしょうか。**

ADF FacesのJavaScriptは、ユーザーへのフィードバックを即座に返すことができるクライアント側検証の実装に適しています。これは、フォームへのデータ入力でタイプミスをしたことのあるエンドユーザーにとって便利です。いずれにしても、保護を厳重にするためには、クライアント側のJavaScript検証と同じ検証を、ビジネス・サービス・レイヤーまたはビュー・レイヤーでサーバーに対して再度実施する必要があります。

#### ADF Business Componentsのエンティティの検証

エンティティ・オブジェクトは1つ以上のビュー・オブジェクトから参照されるため、エンティティ・レベルに検証を追加すると、アプリケーション全体で一貫性のある検証が実行されるようになります。検証ルールを定義する場合、ADF Business Componentsは宣言的なオプションとプログラム的なオプションの両方を使用でき、これには組込みルール一式が含まれます。また、ライフサイクル・メソッドをカスタム・ベース・クラスで上書きし、汎用機能としてセキュリティを追加することができます。

Oracle ADF BCにあらかじめ組み込まれている宣言的な検証ルールは、十分に充実しているため、通常はほとんどの開発要件に適合します。宣言的な検証を使用するには、エンティティ・エディタでエンティティを開き、メニュー・エントリの「Business Rules」を選択します。

エンティティ・バリデータ、エンティティ属性バリデータ、エンティティ・ライフサイクル・コールバック・トリガーを宣言的に定義するには、Business Rulesエディタを使用します。

エンティティ・バリデータ – エンティティに定義されている検証ルールは、選択したエンティティ行に関する行の現在値が変わったとき、およびトランザクションがコミットされたときに実行されます。エンティティ・レベルの検証のメリットは、1か所ですべての属性を相互検証できることです。

図2に示しているのは、エンティティ検証規則を作成または編集するダイアログが表示されているBusiness Rulesエディタです。図に表示されている検証規則は、エンティティ実装クラスについて公開されるJavaメソッドを参照し、ユーザーがエンティティの属性に入力した内容を許可するかどうかを判断します。

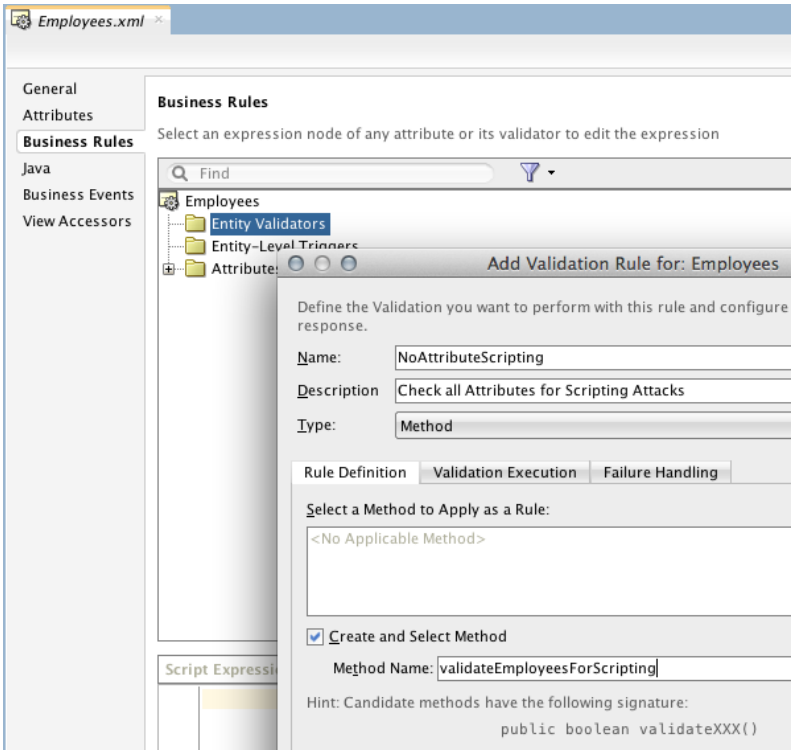


図2：ADF Business Componentsのエンティティの検証

本書の目的に従い、シンプルでありながらも効果的なスクリプティング防止ロジックを見てみましょう。これは、開始と終了を示すマークアップ文字（'<'および'>'）を探す、というものです。

```
public boolean validateEmployeesForScripting() {
  //スクリプトの属性値をチェックします
  Object[] attributeValues =  this.getAttributeValues();
  //'<'、'>'をチェックするためのパターンと、16進でエンコードしたパターン
  Pattern _pattern =
       Pattern.compile("(¥¥%3E)|(¥¥%3e)|(¥¥%3C)|(¥¥%3c)|(<)|(>)");
  Matcher _matcher = null;
  //パターンに一致した場合は文字列のすべての属性をチェックします。
  //最初の一致が見つかった時点でfalseを返します。
  for(Object attributeValue : attributeValues){
    if(attributeValue instanceof String){
      _matcher = _pattern.matcher((String) attributeValue );
      boolean found = _matcher.find();
      if(found){
         //ここでインシデント・レポートを作成します。
          return false;
      }
   }
  }
```

```
  return true;
}
```

起動されたメソッドは、エンティティで使用可能なすべての属性にアクセスし、それぞれのデータ型が`String`であるかどうかをチェックします。文字列型が見つかった場合は、'<'と'>'という文字とこれらのエンコード・バージョンを探す正規表現の文字列に対して属性値をチェックします。このメソッドは、検証が失敗した場合はブール値falseを返し、文字列が検証に合格した場合はtrueを返します。上のコード例では、有効なデータ入力として">"または"<"を使用する必要がある属性値がないことを前提としています。そのような属性値がある場合は、誤検出が発生しないようにするために、正規表現をさらに詳しく定義することが必要になります。一般的に必要な検証メソッドはカスタム・ベース・クラスに定義し、複数のエンティティ定義で宣言的に再利用できるようにすることができます。

なお、上の例ではブラックリスト方式を使用しており、ホワイトリストほど厳密ではありません。サンプルではブラックリスト方式を使用していますが、これはサンプルを簡潔にするためであり、ADF Business Componentsでの正規表現の使用方法を紹介することが主目的であるためです。

**エンティティ属性バリデータ** – エンティティ属性レベルに定義された検証規則は、ユーザー入力を使用してエンティティを更新する前に実行されます。属性レベルの検証は、スクリプティング攻撃に対するチェックとしては、もっとも宣言的かつ一元化されたオプションです。このバリデータは、エンティティ全体の検証の前にコールされます。

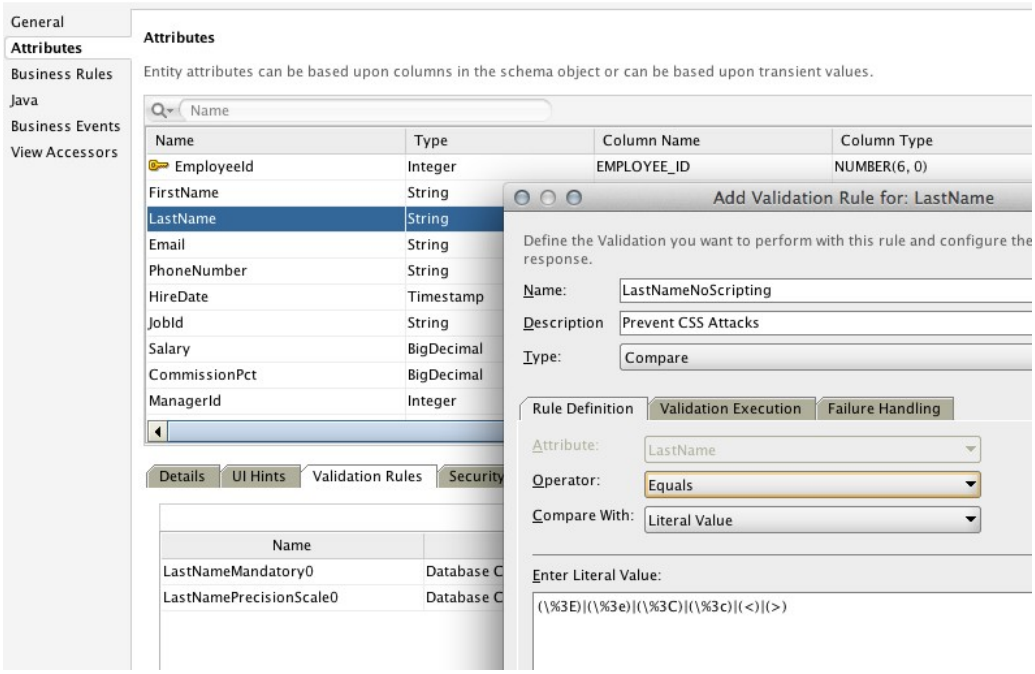


**図3：ADF Business Componentsのエンティティ属性の検証**

図3の属性レベル・バリデータの定義では、エンティティ・レベルのJavaメソッド・バリデータで使用したものと同じ正規表現(\%3E)|(\%3e)|(\%3C)|(\%3c)|(<)|(>)を使用しています。ここでの相違点は、具体的な属性に対してバリデータが定義されていることと、正規表現の照合用として提供されている宣言的なバリデータの1つを使用していることです。ただし、属性にメソッド検証を実装することもできます。

**エンティティ・ライフサイクル・コールバック** – データを作成、変更または削除すると、特定の順序に従ってエンティティが処理されます。この処理順序をエンティティ・ライフサイクルと呼びます。このようなライフサイクルのメソッドの1つが`validateEntity`であり、このメソッドはエンティティ属性が設定または変更されたとき、あるいはエンティティ行が削除されたときにコールされます。カスタム・エンティティ・クラスに含まれる個々のエンティティ、またはカスタム・フレームワークのベース・クラスを使用するすべてのエンティティの`validateEntity`メソッドは、オーバーライドが可能です。たとえば、すべての文字列属性でクロスサイト・スクリプティングが試行されたかどうかをチェックするコードは、カスタム・フレームワークのベース・クラスを使用して追加できるため、同じ検証コードをすべてのエンティティに個別に追加する必要はありません。

#### ADFデータ・コントロール・レイヤーの宣言的な検証

前述のとおり、ADF Business Componentsには、ADFリクエスト・ライフサイクルと緊密に統合された独自の宣言的な検証フレームワークが用意されています。POJO、Web Service、Enterprise JavaBean（EJB）のような他のビジネス・サービスのために、Oracle ADFデータ・コントロールには、コレクションに公開されている属性に開発者が宣言的に検証規則を追加できるオプションが用意されています。開発者は、クロスサイト・スクリプティングのリスクを軽減するために、ADF Business Componentsの場合と同様の方法で正規表現を使用できます。

ADF Business Components以外のデータ・コントロールはすべて、モデル・プロジェクトに配置されている`DataControls.dcx`ファイル（データ・コントロールのレジストリ）に定義されます。`DataControls.dcx`ファイルをダブルクリックしてData Controls Registryエディタを開き、データ・コントロールとそれに含まれるコレクションを確認してください。

コレクションの属性に検証を追加するには、コレクションを選択し、図4に示されている鉛筆アイコンをクリックします。

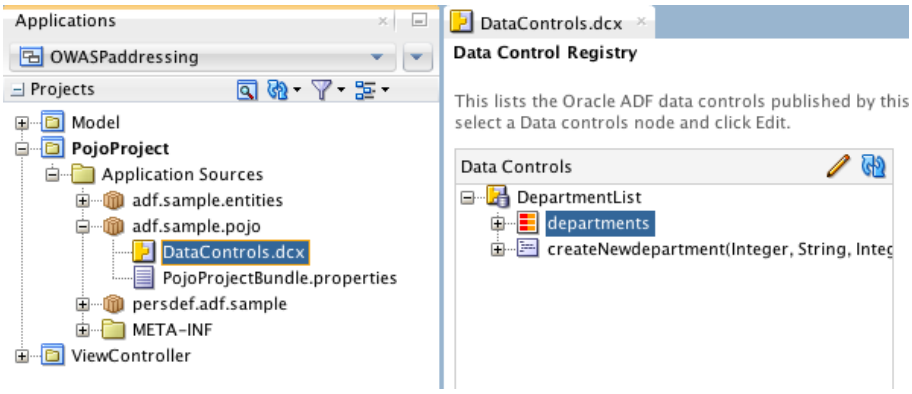


図4：DataBindings.cpxエディタのPOJOデータ・コントロール定義

コレクション・エディタで「Attributes」メニュー・オプションをクリックし、検証を定義する属性を選択します。Validation Rulesタブに切り替え、緑色のプラス記号アイコンをクリックして新しい検証規則を作成します。XSSを防止するために、「RegularExpression」バリデータを選択し、属性が更新されたときに実行する式を追加します。検証が失敗したときに表示するエラー・メッセージも、同じダイアログを使用して定義します。

データ・コントロールに定義した検証規則はすべて、エンティティまたはエンティティの属性にアクセスするビューに関係なく適用されます。そのため、ページに固有の検証でない限り、検証はすべてデータ・コントロールに追加する必要があります。

ヒント：ADFデータ・コントロールでは、公開されるメソッドの引数に検証を定義できません。メソッド・バインディングから作成された入力フォームのフィールドを検証するには、ADFバインディング・レイヤーまたはADF Facesで宣言的な検証を使用します。

#### ADFバインディング・レイヤーでの宣言的な検証

ADFページに作成したADFバインディングに定義されている属性値バインディングに、宣言的な検証規則を直接定義できます。一部のケース（前述したパラメータ・フォームの検証など）ではこのアプローチが役立ちますが、一般的には、ADF Business Componentsのエンティティまたはデータ・コントロールに対してより一元的に検証を適用するほうが合理的です。

#### JSFバリデータ

ADF Facesには、特定の項目の値として許可されているパターンに対してユーザー入力をチェックするための正規表現バリデータがあります。正規表現バリデータの動作タグはJDeveloperのADF Facesのコンポーネント・パネルに含まれ、"Operations"というタグ・カテゴリの下にリストされています（図5）。

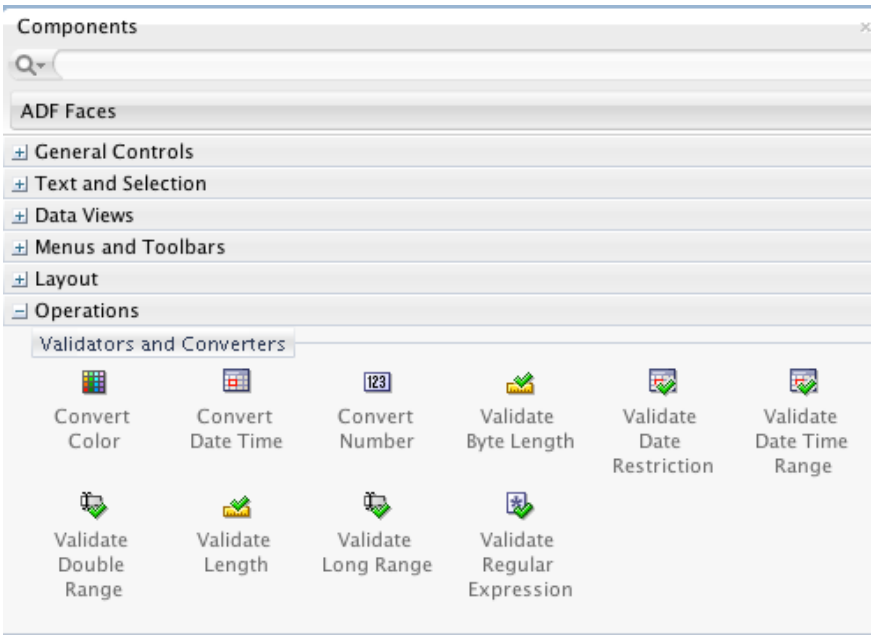


図5：ADF FacesのJSFバリデータ

バリデータは、コンポーネント・パレットからドラッグして入力フィールド・コンポーネントにドロップします。ユーザー・データ入力を検証するJavaの正規表現は、プロパティ・インスペクタで追加することもできます。

ユーザー入力が正規表現に適合しない場合は検証が失敗し、JSFのリクエスト・ライフサイクルはショートカットされるため、モデルの更新は阻止され、無効な入力はビジネス・サービスに侵入できなくなります。

クロスサイト・スクリプティング対策は、スクリーンまたはアプリケーション内の複数の入力コンポーネントで実施することが必要な場合があります。ユーザー入力でのこうしたスクリプティング・インジェクションを防止するために必要なパターンは変わらないため、正しい正規表現を指定するために開発者を煩わせることなく再利用できるカスタムJSFバリデータを作成しておくことを検討する必要があります。

カスタムJSFバリデータを作成する場合は、`javax.faces.validator.Validator`インタフェースを実装するJavaクラスを記述します。

検証ロジック本体はカスタム・バリデータの`validate`メソッドにコーディングし、検証が失敗した場合は`ValidatorException`をスローするようにします。XSSの防止の場合は、おそらく`validate`メソッドでJavaの正規表現を使用してユーザー入力を検証することになるでしょう。

作成したカスタム・バリデータはJSFのfaces-config.xml構成ファイルに登録し、アプリケーション内で使用できるようにします。ADF Facesには、サーバー側のコンポーネント検証の他にもJavaScriptで定義されたクライアント側検証も用意されています。カスタムADF Facesバリデータの記述方法については、Fusion Middleware製品ドキュメントに含まれる『Oracle® Fusion Middleware Oracle ADF FacesによるWebユーザー・インタフェースの開発』を参照してください。

**注：***immediate*プロパティがtrueに設定されたADF Facesのユーザー・インタフェース・コンポーネントには、クライアント側検証が適用されません。*immediate*プロパティが*false*（デフォルト）に設定された他のユーザー・インタフェース・コンポーネントの送信済みの値に、値変更リスナーを使用してコンポーネントからアクセスすると、値はRAW文字列形式で提供されます。*immediate*をtrueに設定したコンポーネント上で値変更リスナーを起動した時点では、他のコンポーネントでの検証およびオブジェクト変換は処理が終了していません。このような場合は、値変更リスナーの中から値にアクセスする必要があるコンポーネントの*immediate*プロパティもtrueに設定されていることを確認するか、Javaが参照しているコンポーネントの`processValidators`メソッドをコールします。

#### JSFコンバータ

Web上でのユーザー入力はすべて文字列としてサーバーに送信され、サーバー上でターゲットのデータ型（数字、日付、同種の型など）に変換されます。反対に、ビジネス・サービスから取得されたデータ・オブジェクトは、ブラウザに表示される前に文字列形式に変換されます。コンバータは、JSFリクエスト・ライフサイクルのコンテキスト内で起動されます。JSFバリデータの場合と同様、カスタム・コンバータを作成できます。たとえば、XSS攻撃、またはSOAPサービスに渡す必要があるコンテンツのXMLエンコーディングから保護するために、ユーザー入力またはデータ出力をエスケープするコンバータなどです。

#### ADF Facesコンポーネントの標準セキュリティ

ADF Facesの出力テキスト・コンポーネントは、デフォルトではセーフ・モードで実行されるため、値はエスケープされます。つまり、文字列内のHTMLマークアップは、解釈するのではなくHTMLを表示するように実行時にソースとしてエンコードされます。そのため、たとえばhelloという値は、helloと太字で表示されるようにマークアップをそのまま送信するのではなく、<b>hello</b>に変換されます。この文字のエスケープ処理は、コンポーネントの"escape"プロパティをfalse設定することで無効化できますが、その前にじっくり検討する必要があります。また、コード・レビューの際はこのようなケースに特に注意する必要があります。

#### カスタムAjaxコールのリスク

ADF Facesにはページ・コンテンツを部分的にリフレッシュしたり送信したりできる機能があるため、ページ全体をリフレッシュせずにサーバーへのリクエストを発行できます。JDeveloper 12cの新しい動作タグ、`af:target`を使用すると、リクエストに対して実行するコンポーネントや応答としてリフレッシュするコンポーネントをさらに細かく制御できます。ADF FacesのJavaScriptクライアント・フレームワークを使用すれば、JSFリクエスト・サイクルのコンテキスト内で動作中のサーバーに対する非同期コールを、JavaScriptで実行することができます。

サーバーに対する独自のAjaxコール（データを問い合わせるためのサーバー側サーブレットのコールなど）を実装する場合は、JSFリクエスト・ライフサイクルが適用されないこと、そのためフレームワークが提供する検証および変換もすべて適用されないことを認識する必要があります。ユーザー入力とサーバーとの通信には、ADF Facesのみを使用することを推奨します。それでもADFアプリケーションでカスタムAjaxコールを使用する必要がある場合は、このアクセス・チャネルを自力で保護する必要があるということを認識する必要があります。JavaScript検証は当てにせず、サーバー側の保護を実装してください。

### OWASP #4 - 安全でないオブジェクト直接参照

オブジェクト直接参照を使用すると、設定した値入力（リクエストURLに含まれるパラメータなど）に基づいてユーザーがドキュメントやデータ・レコードにアクセスできるようにすることができます。オブジェクト直接参照の例としては、データベースからイメージまたはドキュメントを取得するというものがあります（Oracle ADF Fusion Order Demo（FOD）[8]にデモがあります）。

```
<af:image source="/imageservlet?thumbnail=#{row.ProductId}"
          id="i7" shortDesc="#{row.ProductId}"/>
```

FODサンプルでは上のソース例を使用し、データベースにイメージを問い合わせるサーブレットを使用して、データベースにサムネイル・イメージを問い合わせます。このサンプルでは、サーブレット・パスが保護されていないため、匿名ユーザーも認証済みユーザーもここにアクセスできます。ご覧のとおり、このオブジェクト参照を直接ブラウザのURLフィールドで使用して、ページに表示されないイメージをシステムから取り出すのは難しいことではありません。ただし、FODの例の場合はサーブレットからアクセスされる機密データがないため、このアプローチは許容されます。しかし、イメージが製品の写真ではなく、事業報告書や営業報告書などの機密文書だったらどうでしょうか。その場合は、上に示したコードからセキュリティ・リスクが発生します。

URLパラメータ・ベースのアドレッシングとしてよくあるもう1つの例は、ユーザーに代わってサーバーで作成される事業報告書にアクセスし、Adobe PDF形式などのドキュメントとしてクライアントにダウンロードするというものです。この不適切な設計によって、サーバー上の保護されていないフォルダに予測しやすいドキュメント名でドキュメントが保存されることになります。

こうしたURLベースの参照を保護するには、ADF SecurityとOracle Databaseの保護を使用して、Oracle ADFで次のセキュリティ対策を実施します。

[8] http://www.oracle.com/technetwork/jp/developer-tools/jdev/index-092914-ja.html

**保護されているフォルダおよびリソースの使用** – ドキュメントやファイルは保護されているフォルダだけに保存します。ファイル、ドキュメント、データベース・オブジェクトへのすべてアクセスに、認可が必要です。クライアントにドキュメントをダウンロードするときにサーブレットを使用する場合は、web.xmlファイルに定義したセキュリティ制約（またはJava EE 6のServlet 3.0で開発する場合は注釈）を使用してサーブレット・パスを保護します。

**オブジェクト間接参照** – 機密性の高いドキュメントやファイルへのアクセスは、ランダムに生成された名前とドキュメントをマッピングする方法で仮想化します。アクセスできるアプリケーション・スキーマとは異なるデータベース・スキーマのデータベース表を使用することで、仮想ドキュメント名と実際のファイルまたはオブジェクトとの間のマッピングを制御します。たとえば、スキーマはストアド・プロシージャ以外からアクセスできないようにし、認証済みWebユーザーのコンテキスト内で実行されるようにすることができます。オブジェクト参照を仮想化するもう1つの方法としては、セキュリティが適用されているコンテンツ管理システムを使用するやり方があります。

**ファイルまたはドキュメントの参照にランダムな名前を選択する** – 順序やパターンに従ってドキュメント名や参照を作成しないでください。名前はランダムなものにし、予測できないようにする必要があります。

**アクセスの試行を監視する** – ロケーションを保護し、認証を求めるようにしても十分ではないと考えましょう。ユーザーの有効な資格証明がハッカーの手に渡り、システムに侵入したハッカーが機密データを体系的に搾取し始めることもあります。こうした行為を適切に検出して対応できるように、対策を施しておく必要があります。

**データ・セキュリティを適用する** – Oracle Label Securityを使用する場合は、問合せに動的に条件式を追加し、参照を許可されているものだけを認証済みユーザーに表示することができます。ADF SecurityとADF BCを使用すると、View Objectにも同じ機能を実装できます。そのためには、バインド変数を使用して問合せをフィルタするViewCriteriaを作成します（図6を参照）。

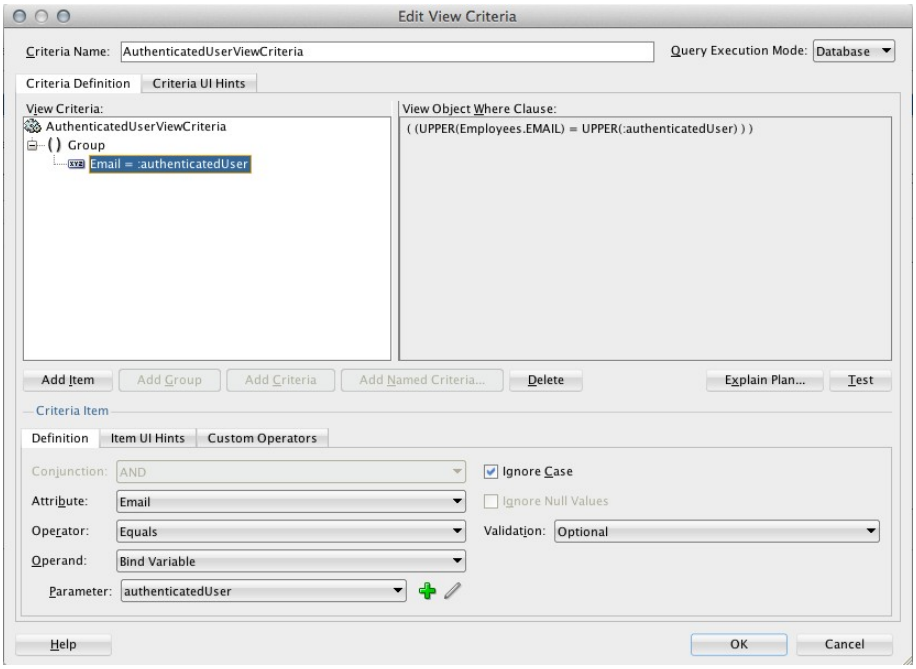


図6：View Objectビューのバインド変数を使用した条件定義

ユーザー名を条件にして問合せが実行されるように、バインド変数の値フィールドでは図7に示すようなGroovy式を使用します。

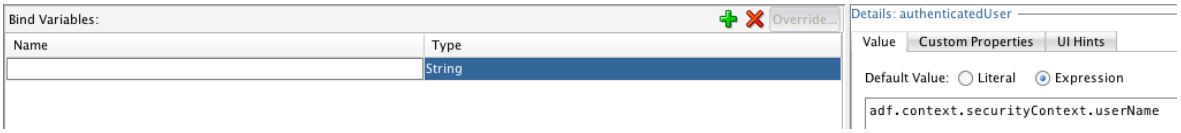


図7：認証済みユーザーにバインド変数からアクセスするためのGroovy式

作成したViewCriteriaは、アプリケーション・モジュール・エディタを使用してViewObjectインスタンスに追加できます（図8を参照）。JDeveloper 12c以降では、実行時に削除できないようにビュー条件を割り当てることができます。

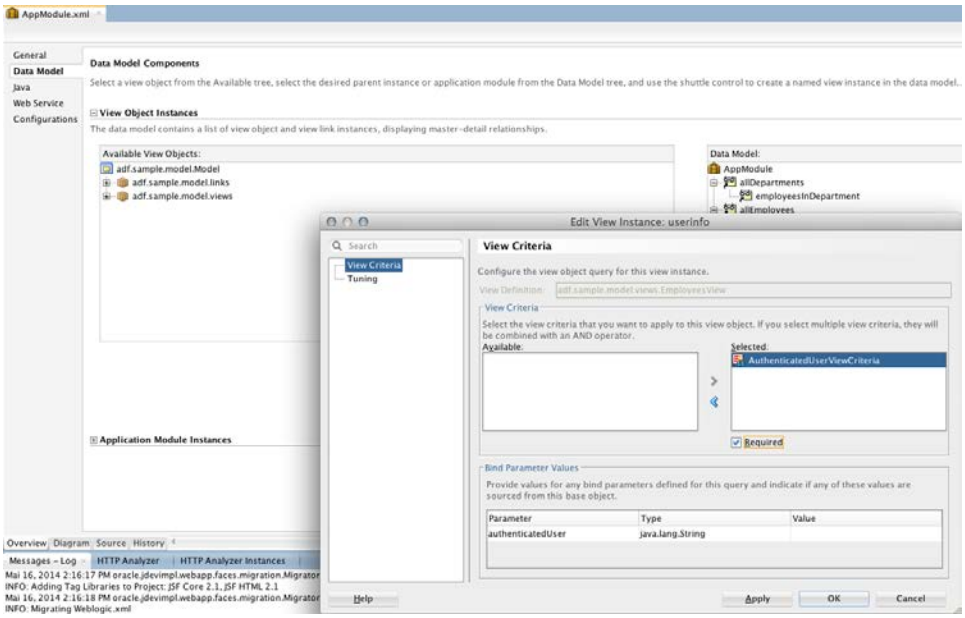


図8：ADF BC View Objectインスタンス・エディタでの必須のビュー条件の割り当て

**注：**JDBC接続またはコンテンツ管理システムからドキュメントや参照オブジェクトに直接アクセスする場合は、引き続き認証済みユーザーにアクセスしてこの情報を問合せとともに渡すことができます。

**ドキュメント参照を期限切れにする –** 機密性の高いドキュメントやオブジェクトへの参照は、作成してから適度な期間が経過したら期限切れにする必要があります。

## OWASP #5 - セキュリティの構成ミス

セキュリティの構成ミスはデータベース、Java EEサーバー、ネットワークを含むあらゆるレイヤーで起こりえますが、アプリケーションそのものでも起こります。本書では、ADFに関連しない構成はすべて対象外とし、ADFがセキュアな環境で実行されるようにする手順が整っているものと想定します。したがって、セキュリティの構成ミスを扱うこの項では、Oracle ADFについてのみ説明します。

### ADF Securityの構成ミス

ADFでは、ADFアプリケーションで認証および認可が実施されるように、ADF Securityが有効化されています。そのため、アプリケーションに対してOracle JDeveloperのADF Securityウィザードを実行すると、図9に示すセキュリティ設定を使用してADF構成ファイル（adf-config.xml）が更新されます。

```
<sec:JaasSecurityContext initialContextFactoryClass="oracle.adf.share.security.JAASInitialContextFactory"
                         jaasProviderClass="oracle.adf.share.security.providers.jps.JpsSecurityContext"
                         authorizationEnforce="true" authenticationRequire="true"/>
```

図9：認証および認可を実施するADF構成

稼働中のADFアプリケーションのADF Securityが無効化される事態につながる構成ミスは、次の状況で発生することがあります。

**アプリケーションの構成が本番用になっていない** – 開発中は、ADFアプリケーションのテストやデバッグ、特定の問題の診断などのために、ADF Securityをadf-config.xmlファイルで無効化することがあります。セキュリティを再有効化し忘れた場合は、保護されてない状態のアプリケーションが本番環境に不用意に公開されてしまうことがあります。このような事態を回避するには、保護対象のアプリケーションのセキュリティ設定（図8を参照）が正しく構成されていることを確認するための必須項目チェックリストを用意する必要があります。

**不完全なセキュリティ構成** – 開発者は、バインド・タスク・フローに対するセキュリティを入念に構成し、それによって、Oracle ADF内で使用するためにアプリケーションに再利用可能なコンポーネントを提供します。ただし、このようなフローを再利用する場合は、コンシューミング・アプリケーション側でセキュリティを有効にしている場合しか、このセキュリティ構成は適用されません。この要件を認識していない開発者は、こうした再利用可能なアーティファクトでアプリケーションをアセンブルするときに認証と認可を有効化するのを忘れることがあります。セキュアなバインド・タスク・フローについては、デプロイする前にアセンブルしているアプリケーションのjazn-data.xmlファイルに、アプリケーション・アセンブラが追加する必要があるポリシー定義とともにセキュリティ要件をドキュメント化する必要があります。

**テスト・ロールの見落とし** – ADF Securityでは、テスト用に作成したデフォルト・ロールはアプリケーションと一緒にデプロイされないようになっています。ただし、セキュリティにとらわれないテストや開発を行うために、開発チームが独自のテスト・ロールを作成し、それに匿名ユーザーIDを付与することがあります。**そのような場合は、テスト・ロールがアプリケーションと一緒に不用意にデプロイされてしまい、本番アプリケーションの実行時の認可は無効化されます。**こうした問題を回避するには、ロールがテスト専用であることを示すネーミング規則を導入し、アプリケーションと一緒にテスト・ロールがデプロイされないように管理者が判断できるようにします。

ユーザーまたはバックドアをハードコードしない - 開発やテスト、不具合診断を簡単にするために、バックドアや特殊なユーザー・アカウントを開発者がシステムに組み込もうとすることがあります。よかれと思って作成したものであっても、このような機能はアプリケーションに甚大なセキュリティ・リスクをもたらします。このようなバックドアが秘密にされ続けることを当てにすることはできず、バックドアが存在するために他のセキュリティ対策がすべて台無しになることもあります。

### 暗黙的なデフォルト

Oracle ADFを含め、すべてのフレームワークにはデフォルト動作があります。たとえばOracle ADFでは、タスク・フローの"URL起動"プロパティおよび"ライブラリ内部"プロパティにデフォルト動作を定義します。これらのプロパティはセキュリティ・トピックと関係があります。

**URL起動** – URL起動プロパティのデフォルト設定は"calculated"です。これは、デフォルト・アクティビティがビューであり、なおかつそのビューで使用するのがJSPXやJSFなどの全ページ・ドキュメント形式であれば、ブラウザのURLフィールドで発行されたhttp GETリクエストからバインド・タスク・フローにアクセスできるという意味です。旧バージョンで実行できるように作成されたコードとの下位互換性をフレームワークで維持するために、このプロパティにはあえてこの値がデフォルトとして設定されています。ただし、セキュリティを最大化するためには、このプロパティを"disallowed"に設定することをお勧めします。

**ライブラリ内部** – JDeveloperのリソース・パレットにバインド・タスク・フローが表示されないようにすることができます。この方法を使用すると、アプリケーション開発者にトップレベルのタスク・フローのみを表示し、サブフローは表示しないようにすることができるため、使用できるAPIを完全に制御できるほか、コンテキスト外のコンポーネントの誤使用を防止することもできます。ライブラリ内部のデフォルト設定はfalseで、これはタスク・フローが表示されることを意味します。

しかし、なぜ上記2つの設定がセキュリティの構成ミスに関する項に書かれているのでしょうか。それは、この2つが、フレームワークのメタデータにデフォルト設定が明示的に示されない例だからです。アプリケーション開発者が設定について知らない場合、またはデフォルト値について異なる理解をしている場合は、ミスを犯す可能性があります。たとえば、URL起動のデフォルト設定は"disallowed"だと開発者が思っていた場合は、そのつもりはなくてもタスク・フローページへの直接アクセスが可能なままになり、アプリケーション・フローのロジック部分が迂回される恐れがあります。このような場合は、エラーになったり（自らハッカーに情報を漏えいしてしまう恐れがあります）、認可していないデータや関数に直接アクセスされてしまったりする可能性があります。そのため、暗黙的なデフォルトが正しい値だと思う場合でも、セキュリティに影響すると判断した機能の値はすべて明示的に設定することをお勧めします。

フレームワークのプロパティに明示的なデフォルトを設定することは、コードのセキュリティ・レビューや後々のメンテナンスの際の構成の解読および理解を容易にすることでもあります。

#### ADF Facesのバージョンの出力

Oracle ADF Facesには、ADFおよびJDeveloperのバージョン情報をページ出力に表示するためのweb.xmlコンテキスト・パラメータがあります。`oracle.adf.view.rich.versionString.HIDDEN`パラメータの値をfalseに設定すると、ADF FacesページのHTMLにバージョン情報が出力されます。本番デプロイメントに関する重要な情報の漏えいを防止するために、パラメータ値は必ずtrueに設定してください。

注：このリスクについては、後ほど*OWASP #9 - 既知の脆弱性を持つコンポーネントの使用*の項で詳しく説明します。

#### プロジェクトのステージ

ADF Facesのバージョン番号の設定に関する項と関連しますが、アプリケーションを本番環境にデプロイするときはweb.xmlのjavax.faces.PROJECT_STAGEパラメータの値を必ず"Production"に変更するようにしてください。こうすることで、バージョン番号が表示されなくなるだけでなく、パフォーマンスが向上するという副次効果も得られます。

## OWASP #6 - 機密データの露出

アプリケーションによってモデル化されたビジネスや国の法律によって当然異なりますが、通常、機密扱いとして分類される可能性のある情報には、職務担当者外秘のデータ、損傷時または損失時のリカバリが困難なデータ、社外秘と見なされるデータがすべて該当します。アプリケーションは、データのリークやパーミッションありでの加工やパーミッションなしでの変更のためにサーフェスがデータ・ストレージ、データ問合せ、データ配信およびデータ操作によって露出されないように設計する必要があります。

機密データの露出はアプリケーション開発時にのみ対処すべきトピックではなく、システム管理者、ネットワーク管理者およびデータベース管理者による管理面からの監視も必要です。本書では、ADFアプリケーション開発者の管理下にあるタスクについてのみ説明します。たとえば、機密データを扱うすべての通信に適用すべき必須の保護であるTransport Layer Security（SSL）などについては本書では取り上げませんので、『Oracle® Fusion Middleware管理者ガイド』[9]を参照してください。

### 認証

データ・アクセスの認可は、ユーザーが認証される環境で実施する必要があります。だだし、ユーザーのなりすましは不可能であるという保証がない場合は、認証のみでは役に立ちません。当然のことながら、ユーザーIDはID管理システムで安全に保管および管理され、強度のあるパスワードを使用し、ログイン・プロセスはSSLで保護する必要があります。

### セッション

デスクトップ・アプリケーションとは異なり、ユーザーはタスクの終了時に必ずWebアプリケーションを閉じるとは限りません。たいていは他のタスクにブラウザ・ウィンドウを再利用するだけであり、クリーンアップが実行されるようにするための"ログアウト"オプションがアプリケーションに用意されていたとしても、使用されない可能性があります。アイドル状態のセッションが長期にわたり維持されないようにすることが重要なのはそのためです。ADFで記述したものも含め、どのJava EEアプリケーションでも、セッション・タイムアウトを`web.xml`構成ファイルで設定します。

ADFセッションはWebセッションで、有効期間のデフォルト設定は35分です。機密データを扱うアプリケーションの場合は、この値をさらに小さくすることが理想です。セッションが早く終了するようにするために、Oracle ADFでできる対策例を次に紹介します。

**ピラー・アーキテクチャを使用する** – ピラーとはADFアーキテクチャの1パターンであり、ここでは論理アプリケーションが複数の独立した小型のJava EEアプリケーションに分割されています。機能の分割の一環として、特に機密性の高いデータや操作へのアクセスを分割して別々のモジュールにすることができ、これにより、それぞれのセキュリティ要件に従ってアプリケーションのパートごとにセッション・タイムアウトを設定することができます。

ピラー・アーキテクチャについて詳しくは、"Oracle ADF Architecture Fundamentals"[10]を参照してください。これは、ADF開発チームが公開している教育ビデオで、さまざまなADFアプリケーション・アーキテクチャ・スタイルについて説明しています。

[9] http://docs.oracle.com/cd/E28389_01/core.1111/b60984/toc.htm

[10] https://www.youtube.com/watch?v=toEuQvp73h8

**ADF Facesセッション・タイムアウト警告を使用する** – アプリケーションを設計する際に考えられる懸念事項として、時間のかかるタスクをユーザーが実行している最中にWebセッションが期限切れになる可能性があげられます。この可能性に対応するには、`web.xml`を変更して長い、ほとんど非現実的なセッション有効期限を設定する代わりに、セッション・タイムアウト警告機能を使用します。ADF Faces固有のコンテキスト・パラメータ[11]を使用すると、警告ダイアログのボタンを押してセッションを延長するチャンスをユーザーに提供することができます。ユーザー・セッションが期限切れになった場合は、セッションが中止されたことを伝える通知がユーザーに送信されます。通知を受けたユーザーはページをリフレッシュし、認証ページを再度開くことができます。こうすると、作業内容を失わせてユーザーを苛立たせることなく、現実に即した短いセッション有効期限でアプリケーションを実行することができます。

**ユーザー・セッションを維持し続けるクライアント側操作をなくす** – サーバー側のデータ更新をクライアントで自動的にチェックするアプリケーション・ロジックが必要になることがあります。ADFの`af:poll`は、これを実現できる便利なコンポーネントですが、pollからサーバーに定期的にpingを行うと、ユーザーがスクリーンを離れ自発的にアプリケーションを操作していなくても、ユーザー・セッションが期限切れにならなくなります。このような場合にユーザー・セッションが期限切れになるようにするには、ユーザーの活動状況に応じて`af:poll`コンポーネントもタイムアウトするようにします。そのために、`af:poll`コンポーネントでは、pollへの割込み設定ができる"timeout"プロパティが公開されています。"timeout"プロパティでは、このコンポーネントの各インスタンスのタイムアウトを設定できるため、ユースケースに応じて異なるタイムアウを設定できます。web.xmlのパラメータ`oracle.adf.view.rich.poll.TIMEOUT`は、timeoutプロパティが設定されていない`af:poll`コンポーネントにグローバル・タイムアウト値を設定するときに使用できます。pollが期限切れになるまでのこのパラメータのデフォルト設定は10分です。`af:poll`コンポーネントについて詳しくは、ADF Facesタグに関するドキュメントを参照してください[12]。

#### 問合せ条件

機密データの機密状態を維持するもっともよい方法の1つは、"知る必要がある"場合にのみデータベースからデータを取得するようにすることです。この方法のほうが、すべてのデータを中間層にフェッチしてからフィルタ処理を行ったり現行ユーザーにはアクセス権がないと思われる情報を編集したりするよりはるかに安全です。こうした制御を実施するための戦略としては、ユーザーIDまたはロール・メンバーシップに基づいて問合せをフィルタする方法や、ミドルウェアではなくデータベースで演算処理を行う方法があります。

ADFの問合せ条件は、Oracleデータベース・ラベル・セキュリティ（別名、仮想プライベート・データベース（VPD））、およびADF Business Componentsのビュー・オブジェクトに適用したビュー条件を使用して実施できます。

**仮想プライベート・データベース** – VPDを使用する場合は、ユーザーが発行したすべての問合せにユーザー固有のwhere句をデータベースで自動的に追加する方法で保護が行われます。Oracle ADFでは、アプリケーション・モジュールの`prepareSession`メソッドをオーバーライドして、VPDによって使用されるコンテキスト情報を設定するため、認証済みのWebユーザーが発行した問合せは、そのユーザーのIDに基づいてフィルタされます。コンテキスト情報を設定するには、Javaメソッドでprepared文を使用して、データベースに格納されている、条件設定を行うPL/SQLプロシージャをコールします。

[11] http://docs.oracle.com/cd/E28389_01/web.1111/b52029/ap_config.htm#ADFUI574

[12] http://docs.oracle.com/cd/E28280_01/apirefs.1111/e12419/tagdoc/af_poll.html

**ビュー条件 –** ADF Business Componentsのビュー条件は、宣言的またはプログラム的にADF BCのビュー・オブジェクト・インスタンスに適用できます。この場合、条件はwhere句として適用されます。ビュー条件はバインド変数を使用してパラメータ化することができます。そのため、この値をGroovyスクリプトで取得することができます。

図7の`adf.context.securityContext.username`は、ビュー条件フィルタで使用する認証済みユーザーの名前を取得するスクリプトです。条件がユーザー名以外の場合は、Javaでユーザー名にアクセスするメソッドをビュー・オブジェクトの実装クラスで公開し、バインド変数の値を判定したり返したりできます。Javaでユーザー名を取得する場合は、次のコードを使用できます。

```
ADFContext adfctx = AdfContext.getCurrent();

String username = adfctx.getSecurityContext().getUserName();
```

**注：** JDeveloper 12.1.3の新機能を使用すると、実行時にビュー・オブジェクト・インスタンスから削除できないようにビュー条件を構成することができます。このようなIDベースのフィルタ処理には、この種の条件が最適です。

#### UIの保護

Oracle ADFでは、ADF Facesを使用してアプリケーションのWebユーザー・インタフェースをレンダリングします。ADF FacesのベースはJavaServer Facesです。つまり、ビューのサーバー側オブジェクト表現とクライアント側表現がHTMLおよびJavaScriptで保持されているということです。

ADFには、ユーザー・インタフェース・レイヤーでデータ・アクセスを防止する方法がいくつかあります（このレイヤーのセキュリティはすべて、モデル・レイヤーおよびデータベースに対するセキュリティ・チェックに追加するべきものであり、置き換えるものではないことに注意してください）。ADF Facesコンポーネントのrenderedプロパティをfalseに設定すると、UIコンポーネントはレンダリングされなくなるため、コンポーネントのデータは決してクライアントに配信されず、偽造したHTTP POSTリクエストでこれらのコンポーネントに更新をかけようとしてもフレームワーク側ですべて拒絶されます。レンダリングは、2つの標準的な方法で動的に制御できます。

- 認証済みユーザーのパーミッションまたはアプリケーション・ロール・メンバーシップをチェックするADF Security式を使用して、特定の情報へのアクセスを制御する方法。ADF FacesコンポーネントのrenderedプロパティにEL式を追加して、レンダリングされたページにコンポーネントを追加するかしないかを制御できます。なお、前の文で"レンダリングされたページにコンポーネントを追加する"と書きましたが、Oracle ADFを使用してアプリケーションを構築している開発者に限らず、アプリケーション開発者にとって重要な設計原則は、セキュリティを仕様としてデフォルトで実装することです[13]。アプリケーションのセキュリティが無効化されても機密データがリークしないようにするには、デフォルト動作として機密データを公開するコンポーネントはレンダリングされたページに追加しないようにする必要があります。
- 実行時にパーソナライズする方法は、機密データをレンダリングするページを制御するもう1つのオプションです。これは、Oracle Metadata Services（MDS）を使用することで実現できます。Oracle ADFでは、あらかじめシードされたカスタマイズをアプリケーションのページおよびビューに定義するときにMDSを使用します。renderedプロパティについての説明で

[13] 『Secure Coding: Principles and Practices』ISBN-13: 978-0596002428

すでに触れましたが、ADF Securityのパーミッションとユーザー・アプリケーション・ロール・メンバーシップは、レンダリングされたページまたはビューにUIコンテンツを追加する前にチェックできます。これを制御するには、カスタム・カスタマイズ・クラスを構築し、下に示すサンプルと同様の方法でADF Securityに対してチェックを行います。このときも、失敗した場合は、セキュリティの実装にカスタマイズを使用するのであれば、コンテンツを削除するのではなく、常にコンテンツをページに追加する必要があります。

下に示すサンプル・コードでは、カスタマイズ・クラスでユーザーのパーミッションをチェックして、レンダリングされたページにUIコンテンツを追加する前にデフォルトの構成オプションをオーバーライドします。そのために、カスタムADF Securityリソース・パーミッションを使用しています。

```
public static final String ACTION = "override";
public static final String RESOURCE_NAME = "adf.sample.cc.Site";
public static final String RESOURCE_TYPE = "Customization";

private boolean isAllowedOverride(){
  boolean hasPermissionGranted = false;
  ADFContext adfCtx = ADFContext.getCurrent();
  SecurityContext securityCtx = adfCtx.getSecurityContext();
  ResourcePermission sitePermission = null;
  sitePermission = new ResourcePermission(RESOURCE_TYPE,RESOURCE_NAME
                      ,ACTION);
  hasPermissionGranted = securityCtx.hasPermission(sitePermission);
  return hasPermissionGranted;
}
```

Oracle MDS内でADF Securityを使用する方法を説明したサンプルの全体と関連ドキュメントは、ADF Code Corner[14]というWebサイトでサンプル"031"として公開されています。

**注：**Metadata Servicesについて詳しくは、Oracle Technology Network（OTN）で公開しているホワイト・ペーパー『Building Customizable Oracle ADF Business Applications with Oracle Metadata Services (MDS)』を参照してください[15]。

[14] http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html

[15] http://www.oracle.com/technetwork/developer-tools/jdev/adfmds-128339.pdf

## OWASP #7 - 機能レベルのアクセス制御の欠落

多重防御の設計パターンでは、複数のセキュリティ・レイヤーをアプリケーションに実装するよう指定されています。これは、基盤となるデータ・オブジェクトがエンティティ・セキュリティによって保護されるとしても、メソッドや操作を実行するアプリケーション機能は認可のチェックによりガードする必要があることも意味しています。たとえば、ビジネス・サービスのメソッドの場合は、JDBCのprepared文を発行してデータベース上の操作を直接実行することでエンティティ検証を迂回できるため、このようなメソッドの起動は認可チェックを使用して保護する必要があります。

同様に、リモートのSOAPサービスまたはRESTサービスをアプリケーションからコールしてデータやアプリケーション状態の情報を更新できますが、このようなコールはADF Securityのデフォルトでは保護されないため、使用する場合は保護することを検討する必要があります。

当初、設計したコンテキスト内でのみメソッドがコールされるとは決して思わないことがベスト・プラクティスです。データを操作する機能へのアクセスはすべて、エンティティでのアクセス制御によるか、ADF SecurityまたはJAASのパーミッション・チェックを使用してメソッドの起動をガードする方法により、保護する必要があります。

ADF Securityには、関数コール（およびユーザー・インタフェース・コンポーネント）を不正使用から保護するためのオプションとして、次のタイプが用意されています。

- **カスタム・リソース・パーミッション** – 関数レベルのセキュリティは、アプリケーション内で実行されるハイレベルのアクション（エンティティを更新するだけとは限らない発注操作など）と関係します。こうした関数の保護をカプセル化できるように、ADF Securityでは、フレームワークが提供する既存のパーミッション以外にカスタム・パーミッションを作成できるようになっています。保護する必要があるメソッド・コールは、続行する前にカスタム・パーミッションと比較してチェックすることができます。
- **ADF Security EL式** – ADF SecurityにはEL式が用意されています。ユーザー・インタフェース・コンポーネントのrenderedプロパティ、またはタスク・フローのルーター・アクティビティでこれを使用して、操作およびナビゲーションが未認可ユーザーに表示されないようにすることができます。実行に必要な権限がないということがすぐに判明する操作の実行をユーザーに促すようなものは、削除することが重要です。

Oracle ADFは、こうした機能を元にして、機能面のセキュリティを効果的に管理できる十分なツールを提供しています。

**注：**この項では、式言語およびGroovyを使用してJavaから認可をチェックするためのADF Securityのオプションについて詳しく説明します。ただし、紙面の都合上、すべてを漏れなく説明することはできないので、Oracle ADFの製品ドキュメント、Oracle ADF Insider[16] Website、Oracle Magazine[17][18]を参照することをお勧めします。

[16] http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adfinsider-093342.html

[17] http://www.oracle.com/technetwork/issue-archive/2012/12-jan/o12adf-1364748.html

[18] http://www.oracle.com/technetwork/issue-archive/2012/12-may/o32adf-1577987.html

### カスタム・リソース・パーミッション

ADFカスタム・リソース・パーミッションは、名前、カスタム・リソース・タイプ、リソース・タイプの定義から選択した1つ以上のアクション、grant文を使用して定義します。カスタム・リソース・タイプは一連のリソース・パーミッションの設計図であり、その定義には名前、物理的なJavaパーミッション・クラスおよび、アプリケーション・リソースや関数の保護に使用するアクションのリストを使用します。パーミッション・クラス（oracle.security.jps.ResourcePermission）は常に同じであり、OPSSから提供されます。

前述した顧客注文の例の場合は、*create*、*delete*、*ship*、*cancel*、*view*および*showStatus*を保護対象アクションとして定義するカスタム・リソース・タイプMyOrdersを作成できます。

カスタム・リソース・タイプを作成するには、ADF Securityリソース・エディタを使用します。このエディタは、JDeveloperのメニューから「Application」→「Secure」→「Resource Grants」の順に選択すると起動します。*ResourceType*フィールドの横にある緑色のプラス記号アイコン（図9）をクリックしてJDeveloperの*Create Resource Type*ダイアログを開き、カスタム・パーミッション・タイプおよび保護するアクションを定義します。

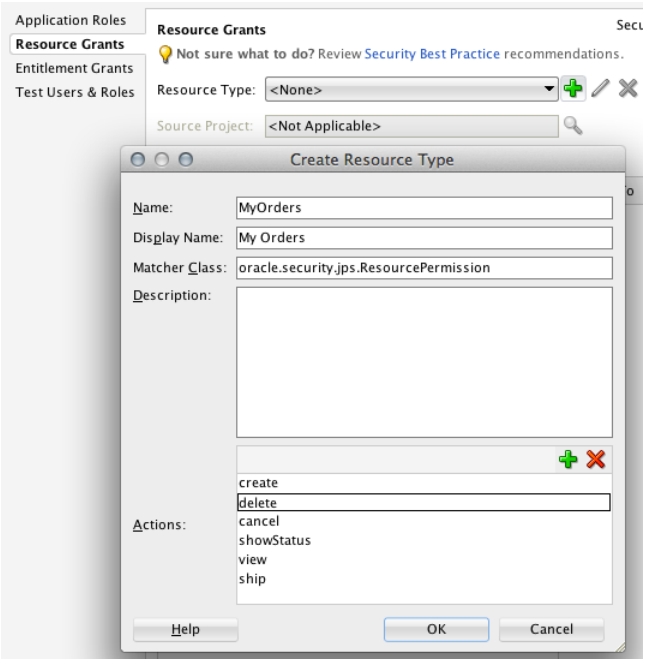


図9：カスタム・リソース・タイプMyOrdersの作成

図10は、別のリソース・パーミッションで*MyOrders*タイプがどのように参照されているかを、grant文およびこの文が対象とするアクションと併せて示しています。

リソース・パーミッションを作成するには、ADF Securityエディタの*Resources*ラベルの横にある緑色のプラス記号アイコンをクリックします。リソース・パーミッションは、名前、パーミッションを付与する1つ以上のアプリケーション・ロール、参照するカスタム・リソース・タイプで公開されるアクションの選択肢を使用して定義します。

図10に示す"shipOrder"パーミッションは、"OrderEmployee"アプリケーション・ロール（ADFアプリケーションで作成したカスタム・アプリケーション・ロール）に付与されます。アプリケーション・ロールはデプロイメントの後処理中にエンタープライズ・ロール（ユーザー・グループ）にマップされ、ID管理システムに定義されているユーザーおよびグループからアプリケーション・セキュリティが抽出されます。

実行時は、式言語またはコードでADF Securityを使用して認証済みユーザーのカスタム・リソース・パーミッションをチェックすることで、エンティティ、オブジェクトおよび関数のセキュリティを適用することができます。

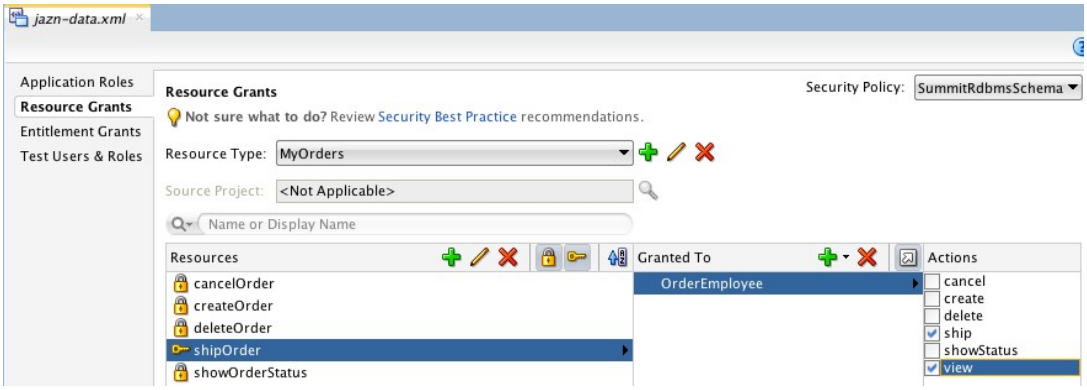


**図10：ユーザーにパーミッションを付与するためのコンテナとしてのリソース作成**

### ADF SecurityContext

OPSSの`SecurityContext`インタフェースには、認証済みユーザーの`Principal`やJAASの`Subject`といったセキュリティ関連情報にアクセスするとき、またはJavaやELから認可チェックを実行するときにアプリケーション開発者が使用できるプログラミングAPIを定義します。

ADF Securityコンテキストによって公開される重要なメソッドの代表的なものは次の通りです。

- getUserPrincipal – 認証済みユーザーのプリンシパル・オブジェクトを取得します。
- getUsername – 認証済みユーザーのユーザー名（ユーザーが未認証の場合は"anonymous"）を取得します。
- hasPermission（パーミッション） – 認証済みユーザーに特定のパーミッションが付与されている場合はtrueを返します。
- isAuthorizationEnabled – 認可を有効にしてADF Securityが構成されている場合はtrueを返します。

- isAuthenticationEnabled – 認証が必須にしてADF Securityが構成されている場合はtrueを返します。
- isAuthenticated – ユーザー認証が成功している場合はtrueを返します。
- isUserInRole（文字列） – ユーザーが特定のエンタープライズ・ロールまたはアプリケーション・ロールのメンバーの場合はtrueを返します。

`SecurityContext`オブジェクトにJavaからアクセスする場合は、ビュー・レイヤーまたはADF BCビジネス・レイヤーで下に示すコード行を使用します。

```
SecurityContext securityContext =
                ADFContext.getCurrent().getSecurityContext();
```

ユーザー・インタフェース・コンポーネントに追加した式言語、またはタスク・フロー・ビューまやルーター・アクティビティで使用する式言語からセキュリティ・コンテキストにアクセスする場合は、次のコードを使用するだけです。

```
#{securityContext.…}
```

セキュリティ・コンテキストにアクセスする3つ目の方法は、Oracle ADF Business ComponentsからGroovy式を使用するやり方です。

```
adf.context.securityContext
```

**注：**28ページの、"OWASP #3 - クロスサイト・スクリプティング（XSS）"の項の図6は、アプリケーション・セキュリティでGroovyを使用する方法の例を示しています。

#### ADF Security EL式

ユーザー・インタフェース上での認可チェックを簡素化するために、ADF SecurityにはEL式の特殊なセットが用意されています。これらを使用すると、ナビゲーション・コマンドのrenderedプロパティ上などで認可チェックを実行できます。

別の例としては、ユーザーの認証状態に応じて機能の表示/非表示をアプリケーション側で制御するというものがあります。

たとえば、オンラインの航空券予約ポータルでは、チケットを予約するにはログインする必要があることを示すテキスト・ボックスを表示できます。ユーザー認証が済んだら、テキスト・ボックスはADF Security ELを使用して削除します。

このユースケースで使用できるADF Facesコンポーネントは`af:switcher`[19]です。

`af:switcher`コンポーネントは、スイッチャーの*facet*プロパティに設定されている式の結果に基づいて、カスタム・ファセット領域の表示/非表示を変更します。このコンポーネントの*defaultFacet*プロパティには、EL式の結果に一致するファセット名がない場合にレンダリングすべきファセットを定義します。前述した航空券予約ポータルの例の場合は、未認証の場合に表示するテキスト・ボックスを保持している*defaultFacet*プロパティの値を、ファセット名に設定します。

**注：**機密情報を含むコンポーネントは非表示にするのではなく、コンポーネントの*rendered*プロパティに記述したADF Security ELを実行してJSFコンポーネント・ツリーから削除する必要があります。

[19] http://docs.oracle.com/middleware/1212/adf/TROAF/tagdoc/af_switcher.html

使用できるADF Security式を下のリストに示します。

*#{securityContext.authenticated}*

ユーザーが認証されているかどうかに基づいてtrueまたはfalseを返す式。

*#{securityContext.userName}*

ログイン時にユーザーが入力した名前を取得する式。

*#{securityContext.userInRole['roleList']}*

認証済みユーザーが、リストしたいずれかのセキュリティ・ロールのメンバーである場合に、trueを返す式。

*#{securityContext.userInAllRoles['roleList']}*

認証済みユーザーが、リストしたすべてのセキュリティ・ロールのメンバーである場合に、trueを返す式。

*#{securityContext.taskflowViewable['target']}*

ターゲットとして定義されているタスク・フローのユーザー権限に基づいて、trueまたはfalseを返す式の文字列。ターゲットは、WEB-INFディレクトリ名を含めたタスク・フロー定義ファイルの名前とIDです。

例：
```
#{securityContext.taskflowViewable
['/WEB-INF/EmployeeUpdateFlow.xml#EmployeeUpdateFlow']}
```

*#{securityContext.regionViewable['target']}*

ページに関連付けられているADF PageDefファイルを表示するためにユーザーに付与されているパーミッションをチェックする式。PageDefファイルは、このファイルの名前とパッケージ名を"target"引数として参照します。

例：
```
#{securityContext.regionViewable
['myorg.adf.sample.views.pageDefs.EmployeesPageDef']}
```

*#{securityContext.userGrantedResource['permission']}*

カスタム・リソース・パーミッションに定義されているアクションに対するユーザーのパーミッションをチェックする式。"permission"引数には、リソース・パーミッションの名前、リソース・タイプおよび認可するアクションの名前をセミコロンで区切ったリストがまれます。

例：
```
#{securityContext.userGrantedResource[
'resourceName=shipOrder;resourceType=MyOrders;
 action=ship']}
```

*#{securityContext.userGrantedPermission['permission']}*

ADFアプリケーション以外のJava EE Webアプリケーションが組織で使用されていることがあります。そのような場合は、既存のJavaパーミッション・クラスが存在

し、ADFアプリケーション内でそれを再利用することが必要となることがあります。userGrantedPermission式を使用すると、ADFの開発者はこのようなパーミッション・クラスに対する認可をチェックできます。ELコールの"permission"引数には、物理的なパーミッション・クラス、ターゲット、および認可の実行対象となるアクションを定義します。

*#{securityContext.userGrantedPermission* ['permissionClass=<class>; target=<target>; action=<action>']}

たとえば、ユーザーがADF BCエンティティ（例："Orders"）を削除できない場合にコマンド・ボタンを表示しないようにするには、コマンド・ボタンの*rendered*プロパティに次のELを追加します。

*#{securityContext.userGrantedPermission* ['
permissionClass=oracle.adf.share.security.authorization.EntityPermission;
target=adf.sample.model.entities.Orders;
action=delete']}

#### ADFのプログラム的セキュリティ

ADF Securityのパーミッションはすべて実行時のJavaクラスで表されます。このクラスは、アプリケーション・コードでインスタンス化して動的にチェックすることができます。よく使用されるクラスは次のとおりです。

- カスタム・リソース定義用の`oracle.security.jps.ResourcePermission`
- バインド・タスク・フロー用の`oracle.adf.controller.security.TaskFlowPermission`
- アンバインド・タスク・フローのページ・パーミッション用の`oracle.adf.share.[…].RegionPermission`
- エンティティ用の`oracle.adf.share.[…].EntityPermission`
- エンティティ属性パーミッション用の`oracle.adf.share.[…].EntityAttributePermission`

下のコード例は、JSFコマンド・コンポーネントのアクション・リスナー経由で起動されたアクションをガードする方法を示しています。

```
public void onCancelOrder(ActionEvent actionEvent) {
 //ADFContextクラスからADF Securityコンテキストを取得します
 SecurityContext securityCtx =
                ADFContext.getCurrent().getSecurityContext();
 //前もってorders用に作成しておいたResourcePermissionのインスタンスを
 //作成します。なお、このパーミッションはエンティティに対しては使用されません。
 //認証済みユーザーが注文をキャンセルできるかどうかを定義するパーミッションであるため、
 //プログラム・ロジックで使用します:ResourcePermission(String rtype,
```

```
  //String name, action)
   ResourcePermission resourcePermission =
           new ResourcePermission("MyOrders","cancelOrder","cancel");
  //ユーザーにパーミッションが付与されているかどうかをチェックします
   boolean userHasPermission =
                       securityCtx.hasPermission(resourcePermission);
  //ユーザーにパーミッションが付与されている場合は、関連付けられているロジックを実行します
   if (userHasPermission){
       //権限付きロジックをここで実行します
   }
   //ユーザーに権限が付与されていない場合は、アクションが試行されたことをログに記録し、UI
   の改良（認可されていないユーザーにはキャンセル・
   //オプションを表示しないようにするなど）に活用できるようにします
   else{
   // … アクションの試行が失敗したことをログに記録します
  }
}
```

上に抜粋したコードは、“action”パーミッションに対して認可をチェックしています。アプリケーション・メニューに含まれるメニュー項目を“view”パーミッションに対してチェックし、キャンセル・オプションをユーザーに表示すべきかどうかを判断することもできます。後者のユースケースの場合は、前述したADF Security式を使用することになります。

**ヒント：**アプリケーション・モジュールやビュー・オブジェクト実装クラスで公開されるカスタムADF BCクライアント・メソッドはADF Securityで自動的に保護されるわけではありません。カスタム・メソッドを保護するには、前述したとおり、カスタム`ResourcePermission`を使用します。同様に、ADF BCのフレームワーク操作（Create、CreateInsert、Deleteなど）を保護するには、フレームワークのベース・クラスとそのメソッドをオーバーライドし、Javaのパーミッション・チェックを追加します。ADFにバインドされているUIコマンド・コンポーネントのセキュリティを強化するもう1つのオプションとしては、関連付けられているADFメソッド・バインディングや操作バインディングをコマンド・コンポーネントから直接起動せず、マネージドBeanを介する方法があります。Beanを作成するには、JDeveloperビジュアル・エディタでADFバウンド・コンポーネントをダブルクリックします。すると、コンポーネントのアクション・メソッドを作り込むためのマネージドBeanを作成または選択できるダイアログが表示されます。アクション・メソッドを作成する場合は、構成した動作を保存するために、JDeveloperのオプションを使用して、コンポーネントにバインドされているADFバインディング・レイヤーのメソッドまたは操作を起動するJavaコードを生成することができます。メソッド・バインディングまたは操作バインディングの起動を保護するには、生成されたコードの前後に、上に示した例と同様のカスタム`ResourcePermission`に対するチェックを挿入します。

## OWASP #8 - クロスサイト・リクエスト・フォージェリ（CSRF）

クロスサイト・リクエスト・フォージェリとは、フィッシング・サイトと気付かれずに認証済みユーザーにリンクを実行させたり、HTMLコンテンツをビューに注入するクロスサイト・スクリプティングの踏み台として認証済みユーザーを利用したりする方法で、第三者のWebアプリケーション・リクエストによる攻撃が行われるリスクのことです。JavaServer Facesで注意すべきものは、ビューの状態です。

### ページ・トークン

JavaServer Facesでは、ビューの状態でクライアントの状態を追跡し、後続のリクエストがサーバー側ビューの状態を識別し、変更されたデータ値があればそれを特定し更新できるようにしています。ビューの状態を保存するオプションとしては、クライアントの非表示フィールドを使用する方法とクライアント・トークンを使用する方法の2つがあります。

**非表示フィールド** – 非表示フィールドを使用すると、ほとんどのJavaServer Facesアプリケーションでは、base64でエンコードした値としてビューの状態を非表示UIフィールドに保存します。このオプションの場合、ビューの状態が明示的に暗号化されていない限り、クライアント上で状態操作が可能なため、あまり安全な方法とは言えません。

JSF 2.2より前は、こうしたビューの状態の暗号化が可能でしたが、デフォルトでは有効化されていませんでした。JSF 2.2の暗号化は、開発者が選択したアルゴリズムに基づいてデフォルトで有効化されます。ビューの状態の暗号化はweb.xmlで定義します。

**クライアント・トークン** – クライアント・トークンを使用すると、ビューの状態はサーバー上でユーザー・セッションに保存され、トークンを使用するクライアント以外は状態を識別できなくなります。これがデフォルトであり、ADFアプリケーションではこの方法を使用して状態を保存することをお勧めします。トークンは、MyFaces Trinidad状態マネージャを使用してビューごとに暗号化されます。このデフォルト設定のままにすることをお勧めします。

### XSS

CSRF攻撃を軽減するためのもう1つの方策は、クロスサイト・スクリプティングのリスクを監視することです。本書のXSSの項で説明したとおり、ユーザー・セッションが悪用されたりJavaScriptや追加したHTML形式コンテンツを使用した不正なアクションが実行されたりしないようにするには、すべての入力とすべての出力をエンコードする必要があります。

### フレームバスティング

フィッシングとはCSRF攻撃の一種で、クレジットカード会社や銀行のサイトのように大勢の人が利用するウェブサイトを悪意のあるページに"フレーム"として組み込んでユーザーのパスワードや機密データ情報を探り出す手法です。ユーザーは、サイトにこのような手口が組み込まれているとは気付かず、電子メールやドキュメントに記載されたリンクに従い、だまされてこうしたサイトを使用することがあります。その後、悪意のあるページでは、ホスティングされているページ上のユーザーのアクティビティを監視したり情報を捕捉したりできます。関連するフィッシング攻撃には"クリックジャック攻撃"があります。クリックジャック攻撃の場合は、組み込まれたサイトではなくそれを取り囲む悪意のあるページに含まれるボタンをユーザーにクリックさせてコードを実行します。

こうしたフィッシング手法のリスクを軽減するために、ADF Facesには"フレームバスティング[20]"と

[20] http://docs.oracle.com/cd/E50629_01/adf/ADFUI/ap_config.htm#BABDHGEJ

いう機能が用意されています。この機能は、次のいずれかのコンテキスト・パラメータを使用して、ADFアプリケーションのweb.xmlファイルに構成します。

`oracle.adf.view.rich.security.FRAME_BUSTING` – ADF 11gでADFアプリケーションを実行する場合は、このコンテキスト・パラメータを使用します。

`org.apache.myfaces.trinidad.security.FRAME_BUSTING` - ADF 12c以降でADFアプリケーションを実行する場合は、このコンテキスト・パラメータを使用します。

これらのパラメータには両方とも同じ効果があり、アプリケーションをこの方法で別のページにラッピングすることができなくなります。

### OWASP #9 - 既知の脆弱性を持つコンポーネントの使用

ソフトウェア・ベンダーは、自社ソフトウェアで見つかったセキュリティの問題に対応するソフトウェア・パッチを頻繁にリリースします。セキュリティに関連する修正がパッチに含まれている場合は特に、パッチやベンダー更新ができるだけ速やかに適用されるようなアプリケーション・メンテナンス・プロセスを計画することが不可欠です。既知の公表済みの問題が含まれる古いソフトウェア・リリースを使い続けると、脆弱な状態に置かれたままになります。

こうしたパッチ適用が必要とされる好例としては、最近発見された"ハートブリード"[21]と呼ばれるOpenSSLのバグがあります。このバグに伴う脆弱性は2014年初旬に公表されており、同時にパッチがリリースされています。パッチをインストールせずにOpenSSLを使用していた会社は、この既知の脆弱性の存在により、システムをリスクにさらしていました。

この種のセキュリティ脆弱性を軽減するためのもっとも効果的な施策は、Oracle ADFのリリースやOracle Critical Patch Updateプロセスに関する最新情報を早め早めに入手するようにすることです。

#### リスク：JSFのプロジェクト・ステージとADF Facesのバージョン番号

開発時やテスト時に確認することが必要になりそうな情報は、Oracle ADF FacesおよびADFのバージョン、アプリケーション開発に使用しているJDeveloperのビルド番号です。この情報は、次のweb.xmlコンテキスト・パラメータを設定することにより、非表示フィールドとしてADF Facesページに出力することができます。

```
<context-param>
  <description>
    ADF FacesのHTMLページの下部に表示する'Generated by...' コメントにバージョン番号情報を含
      めるかどうかを決定します。
  </description>
  <param-name>
    oracle.adf.view.rich.versionString.HIDDEN
  </param-name>
  <param-value>false</param-value>
</context-param>
```

[21] http://en.wikipedia.org/wiki/Heartbleed

`oracle.adf.view.rich.versionString.HIDDEN`パラメータを上のように"false"に設定すると、バージョン情報がレンダリングされたページに出力されます。本番へデプロイ済みのアプリケーションについては、常にこのパラメータをtrueに設定することを推奨します。

JDeveloper 11g R2および12c以降では、JavaServer Facesの`PROJECT_STAGE`パラメータを使用して、デプロイメント・タイプに基づきADF Facesの`HIDDEN`パラメータを切り替えることができます。デフォルトのデプロイメント・タイプはproductionで、`PROJECT_STAGE`パラメータが設定されていない場合は、`HIDDEN`パラメータがfalseに設定されていても、これは無視されます。ただし、ADF Facesのこの安全なデフォルト動作を使用する場合でも、本番デプロイメントでは常に`HIDDEN`パラメータをtrueに設定するのが最善策です[22]。

**注：**アプリケーションのセキュリティの構成ミスにより、必要のない情報がハッカーに提供されてしまうため、上記のリスクは"*OWASP #5 - セキュリティの構成ミス*"にも該当します。

#### ゼロデイエクスプロイトに対する防御

パッチを最新状態にするよう努力していたとしても、新しい未知の脆弱性、すなわちゼロデイ[23]が攻撃に使用される可能性は常にあります。このような脆弱性攻撃からいつまでも完全に安全なソフトウェアはありません。対策としてできることは、アプリケーションをできるだけセキュアにコーディングし、アプリケーションについて公開する情報量を減らし、特定のプロファイルを持つシステムをハッカーが見つけにくくなるようにする以外にありません。

ゼロデイ攻撃のリスクを減らすには、使用しているソフトウェアのバージョンを公開しないようにしてください（バージョン番号を非表示にする方法については、前の項を参照してください）。また、本番システムで問題が見つかった場合は、バージョンの詳細やエラーをインターネット上で公式に報告せず、カスタマ・サポートを利用してください。

公開の場で情報を共有する必要があるかどうかは常識的に判断し、みずから提供した情報によってみずからのセキュリティが損なわれないようにしてください。情報としてはメジャー・リリース・バージョンだけで十分な場合がほとんどであり、パッチ・セット・レベルの情報は不要です。

**注：**Oracle ADFではJSFまたはMyFaces Trinidadのバージョンをユーザー側で簡単に更新することができません。この部分のパッチはオラクル側でADF Facesに適用する必要があります。

### OWASP #10 - 未検証のリダイレクトとフォーワード

ADF Webアプリケーションでは、JavaServer Facesのポストバック・ナビゲーションを使用して、ブラウザ内またはADFリージョンに公開されているビュー間のナビゲーションを行います。アプリケーション間でナビゲーションが行われる場合（ピラー・アーキテクチャを使用している場合など）、またはADFアプリケーションからFormsアプリケーションをコールする必要がある場合は、http GETリクエストを使用しますが、そうすると、システムを通過する従来のパスを迂回したハッカーが脆弱性攻撃を仕掛けることのできる攻撃面が増えるというリスクが露呈する可能性があります。

[22] https://blogs.oracle.com/groundside/entry/basic_weblogic_deployment_plans_for

[23] http://en.wikipedia.org/wiki/Zero-day_attack

Oracle ADFを使用する場合は、次の対策によりリスクを軽減します。

- 直接GETリクエストを通じて発行または受信したリクエストはすべて、リクエスト元またはターゲットのIDが検証されるようにします。
- 別のJava EEアプリケーションにデータを渡す必要がある場合は、データが加工されない方法で渡すようにし、データの有効性を検証する方法が受信側アプリケーションでわかるようにします。これに対する解決策として考えられるのは、ストアド・プロシージャを介してアクセスされるデータベース表を使用する方法です。このような場合にアプリケーション間で受け渡しを行う必要がある唯一のパラメータは、(有効期限を短く設定して暗号化した) セキュアなトークンです。
- ADFアプリケーションでのナビゲーションにはGETリクエストまたはリダイレクトを使用せず、ポストバック・ナビゲーションのみを使用するようにします。
- ADFアプリケーションへのアクセス・ポイントの数を制限します。これは、バインド・タスク・フローを使用する方法、およびJDeveloperのプロパティ・インスペクタを使用してすべてのタスク・フローの"URL起動"プロパティをdisallowedに設定する方法で容易に実現できます。
- アプリケーションが最初からではなく途中から起動するようにADFアプリケーションをコールする必要がある場合は、URL起動プロパティを"allowed"に設定してADF Securityを有効にしたバインド・タスク・フローを使用します。こうすることで、タスク・フローに対するアクセス制御が確実に実施されるようになり、アプリケーションのサブ機能へのアクセス・ポイントを 1 つにすることができます。
- ブラウザから直接アクセスできるバインド・タスク・フローの場合は必ず、入力パラメータ値をpageFlowScopeのマネージドBeanに保存し、メモリ属性に直接保存しないようにする必要があります。入力値を保持するマネージドBeanのプロパティの"set<Name>"メソッドには、リクエストに設定された入力値をガードおよび検証するコードを追加しておく必要があります。
- アンバインド・タスク・フローのADFページはブックマーク可能として構成できます。ブックマークはビュー・アクティビティ定義上で構成します。ブックマークされたページが、リストアされた問合せの状態を使用して表示されるように、入力パラメータを定義することができます。重要なのは、パラメータ値の妥当性と正しさをチェックするコンバータをすべてのパラメータに定義することです。想定される入力パラメータの型は文字列であるためオブジェクト変換が必要でない場合でも、コンバータを使用してリクエストのパラメータを検証してください。なお、ブックマークの入力パラメータには余分なバリデータ・プロパティがありません。検証にもコンバータを使用する必要があるのはそのためです。

## 境界セキュリティ

セキュアでない環境でアプリケーションそのものを実行するのであれば、アプリケーションのセキュリティは役に立ちません。境界セキュリティとは、ADFアプリケーションの外側にあるサーバー、ネットワーク、および他のデータ・アクセス・チャネルに追加する保護レベルを表すものです。本書で指摘したとおり、2013年版のセキュリティ脆弱性OWASPトップ10のすべてが、アプリケーション開発者に関係するトピックというわけではありません。

Transport Layer Securityや専門的なID管理は、Fusion Middlewareの管理においては明らかにセキュリティ管理者の担当領域に分類されます。本書で概説したセキュアなコーディングを、開発者がベスト・プラクティスに従って施すことができますが、保護されないままになっている領域が他にある場合はこれで十分ではありません。

## どの程度のセキュリティが必要でしょうか

「アプリケーションに必要な保証の程度は、それぞれに特有のリスクの規模と性質のほか組み込める対策のコストとも非常に強く関係します。アプリケーションにどの程度のセキュリティが必要かと問われれば、十分なセキュリティと答えます」

- Mark G. Graff、Kenneth R. van Wyk[24]

David Knoxは著書『*Effective Oracle Database 10g Security by Design*』[25]の中で、セキュリティとユーザビリティとパフォーマンスとの間の相互作用と依存関係を表す三角形を描いています。著者が強調しているのは、セキュリティは単独で存在するものではないということと、セキュリティを厳しくすると、ユーザーはパフォーマンスと使いやすさに関してアプリケーションの捉え方を変えることになるだろうということです。セキュリティが100%保証されるアプリケーションは存在しませんが、ほとんどの場合そのようなものは必要ありません。アプリケーションに必要なのは、アプリケーションで特定されたリスク、および保護する必要があるか保護することが法律で定められているデータの性質に見られるリスクからアプリケーションを保護するのに十分なセキュリティだけです。

本書では、入力の検証をどこに配置するかなど、多数のオプションを紹介しました。また、多重防御の設計パターンにも触れました。ですが、これはつまり、アプリケーションに関わるテクノロジー・レイヤーとビジネス・レイヤーのすべてで入力を検証する必要があるということなのでしょうか。おそらくそのようなことはなく、ある程度まで検討すれば十分でしょう。

では、どの程度であれば十分なのでしょうか。十分かどうかを判断できるのは、OWASPがまとめたセキュリティ脅威のような一般的なセキュリティ脅威について明確に理解したうえでリスク分析に基づいて作成した計画がある場合のみです。ただし、ソフトウェア開発におけるセキュリティ・リスクについてまとめている組織はOWASPだけではありませんので、他のものにも目を通すことをお勧めします。

OWASPの重大なセキュリティ脆弱性トップ10リストは注目を集めているものであるため、本書ではこれについてのみ説明しました。しかしながら、対処が必要なのはこのトップ10だけではありません。たとえば、OWASPトップ10に含まれていない脆弱性には*ソーシャル・エンジニアリング*があります。ソーシャル・エンジニアリングにはさまざまな側面があり、その手法もフィッシングやショルダー・サーフィンをはじめ、情報をユーザーからだまし取るというものまで広範囲に及びます。

要するに、カスタム・アプリケーションを開発する場合に購入できる一番の保護は、教育なのです。教育を実施することで、リスクの特定が可能になり、リスク軽減対策に取り組むことができ、どの程度のセキュリティで十分なのか判断することもできるようになります。

[24] 『Secure Coding:Principles and Practices』、ISBN-13:978-0-596-00242-8

[25] 『Effective Oracle Database 10g Security by Design』、ISBN-13:978-0-07-223130-4

## まとめ

本書では、Oracle ADFを念頭に、2013年版OWASPのセキュリティ脆弱性トップ10リストの関連項について説明しました。また、ブラウザWebアプリケーションやモバイル・アプリケーションを保護する際にセキュリティ意識の高い開発者が利用できるOracle Application Development Framework（ADF）の機能およびツールを紹介しています。

アプリケーション開発者およびプロジェクト・マネージャーにとって重要なのは、どのプラットフォームやフレームワークを使用しても100%のセキュリティが完全な形の機能として提供されることはないと理解することです。セキュリティとは、意識の高さと教育と使用するテクノロジーが組み合わさったものです。Oracle ADFの場合は、アプリケーション開発者がセキュアなWebアプリケーションを記述できるだけのテクノロジーを提供するために最小限必要なツールがすべて揃っています。

## 付録：お勧めの記事

1. OWASP Japan のホームページ

   https://www.owasp.org/index.php/Japan

2. “ADF Architecture Square:ADF Code Guidelines v2.00”

   http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adf-code-guidelines-v2-00-2096456.pdf

3. “Handling the OWASP Top Ten Application Security Risks with Oracle Fusion Middleware”

   http://antonfroehlich.blogspot.de/2012/06/handling-owasp-top-ten-application.html

4. Oracle ADF Security のドキュメント

   http://docs.oracle.com/cd/E57014_01/adf/adf-secure.htm

ORACLE

Oracle ADFのセキュリティ：
OWASPのセキュリティ脆弱性トップ10への対処

2014年10月

著者：Frank Nimphius
共著者：Duncan Mills、Gary Williams、
Denis Pilipchuk

ECCN：EAR99

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

海外からのお問い合わせ窓口：
電話：+1.650.506.7000
ファクシミリ：+1.650.506.7200

oracle.com

Oracle is committed to developing practices and products that help protect the environment



Hardware and Software, Engineered to Work Together