日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2021年3月25日木曜日

SQLインジェクションの対応について

Scott Spendoliniさんが昨年のAPEX@Home 2020にて、以下のセッションを行っています。

APEX Security Checklist
https://www.youtube.com/watch?v=b9esXRx2A-Q

Scott Spendoliniさんは、元々Oracle APEXのプロダクト・マネージャだったのですが、オラクルを辞めてOracle APEXのコンサルティングを行う会社を始めました。そして、昨年オラクルに戻り、今はOracle APEXを使ってアプリケーションを開発するチームを率いています。

2013年に、Expert Oracle Application Express Securityというタイトルで、Oracle APEXのセキュリティに関する書籍も出版されていて、この領域での専門家として認知されています。

このセッションの12:47から20:57まで、SQLインジェクションへの対応が説明されています。

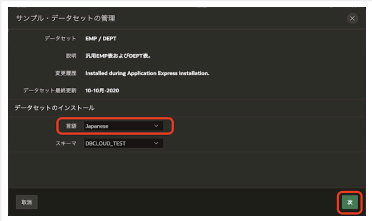
このセッションで説明されている内容を、実際に確認してみました。

準備

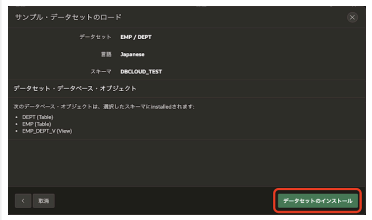
SQLワークショップの**ユーティリティ**から**サンプル・データセット**を呼び出します。

サンプル・データセットの**EMP/DEPT**を**インストール**します。

言語は**Japanese**を選択し、**次**へ進みます。

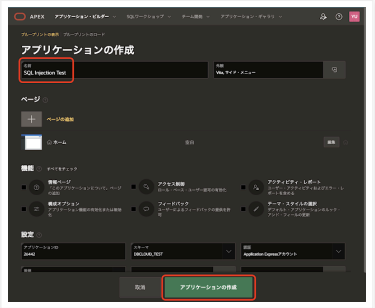


データセットのインストールを実行します。



サンプル・データセットのロードが完了しました。**終了**をクリックします。

続いて空のアプリケーションを作成します。名前は任意ですが、**SQL Injection Test**としました。**アプリケーションの作成**を実行します。

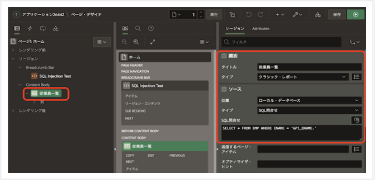


アプリケーションが作成されたら、ホーム・ページをページ・デザイナで開きます。

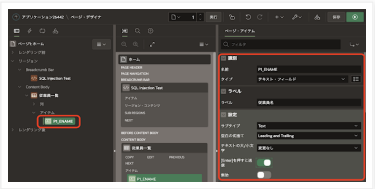
Content Bodyに**タイプ**が**クラシック・レポート**のリージョンを作成します。**タイトル**は**従業員一覧**とし、ソースのタイプをSQL問合せ、SQLとして以下を設定します。

```
SELECT * FROM EMP WHERE ENAME = '&P1_ENAME.'
```

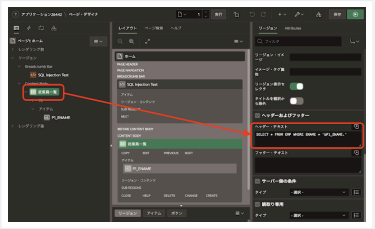
ページ・アイテムP1_ENAMEの値を、置換文字列**&P1_ENAME.**を使って、WHERE句に指定しています。**やってはいけないコーディングの例です。**



ページ・アイテムP1_ENAMEをリージョン従業員一覧に作成します。**名前**は**P1_ENAME**、**タイプ**は**テキスト・フィールド**、**ラベル**は**従業員名**、(送信ボタンを追加するかわりに)**[Enter]を押すと送信**は**ON**とします。



実際に実行されるSQLを確認するために、ソースに指定したSQLを**ヘッダー・テキスト**に指定しておきます。



以上で準備は完了です。これから、なぜこういうコーディングが問題なのか確認していきます。

置換文字列ではなく必ずバインド変数を使う

従業員名に**中島 亜希子**を入力し、レポートを表示します。

レポートを表示するSQLは&P1_ENAME.の部分が置き換えられ、以下のSQLが実行されています。

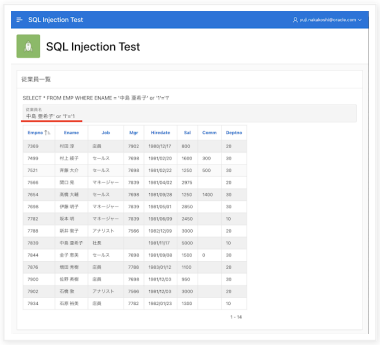
```
SELECT * FROM EMP WHERE ENAME = '中島 亜希子'
```

従業員名を入力している限り、問題は見つかりません。

以下を従業員名に入力してみます。

中島 亜希子' or '1'='1

実行結果は以下になっています。**表EMPの全行が表示されています。**



実際に実行されているSQLは以下です。

SELECT * FROM EMP WHERE ENAME = '中島 亜希子' or '1'='1'

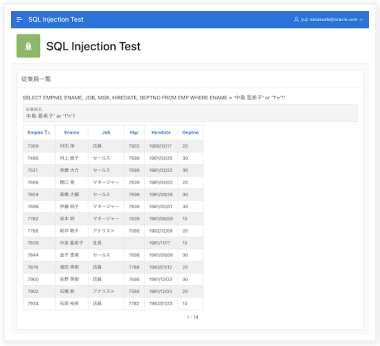
全行選択される条件になってしまっていることがわかります。

従業員のSAL（給与）、COMM（コミッション）はレポートに表示する列から外すことにします。

ソースのSQLを以下に変更します。

SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, DEPTNO FROM EMP WHERE ENAME = '&P1_ENAME.'

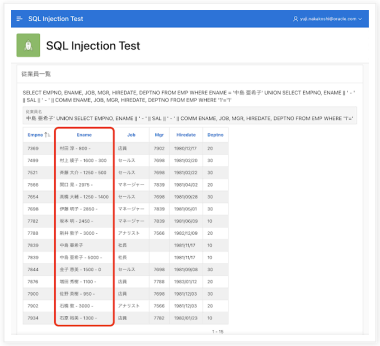
先程の条件であれば、全従業員がリストされます。SALおよびCOMMは表示されません。



従業員名として以下を入力します。

中島 亜希子' UNION SELECT EMPNO, ENAME || ' - ' || SAL || ' - ' || COMM ENAME, JOB, MGR, HIREDATE, DEPTNO FROM EMP WHERE '1'='1

ENAMEとして、**SALとCOMMが表示されている**ことが確認できます。



実行されたSQLは以下です。

SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, DEPTNO FROM EMP WHERE ENAME = '中島 亜希子' UNION SELECT EMPNO, ENAME || ' - ' || SAL || ' - ' || COMM ENAME, JOB, MGR, HI

こうなると、どのようなSELECT文も実行でき、データベースの接続ユーザーにSELECT権限があれば、何でもデータとして取得できる状態です。

例えば以下を従業員名に指定すると、データベースに登録されているユーザーの一覧を取得できるはずです。(ビューALL_USERSのSELECT権限が必要です。)。結果を載せるのは控えます。

中島 亜希子' UNION SELECT null, USERNAME, null, null,null,null from ALL_USERS where '1'='1

従業員名として以下を指定すると、スキーマにどのような表があり、どのような列を持っているかを確認することができます。

中島 亜希子' UNION SELECT null, TABLE_NAME, COLUMN_NAME, DATA_LENGTH,null,null from USER_TAB_COLS where '1'='1

これも結果を載せるのは控えます。

これほど危険なコーディングなので、**SQLに置換文字列は絶対に使うべきではありません。**

SQLを記述する際には置換文字列ではなく、必ずバインド変数を使います。

ページ１のコピーとしてページ２を作成し、ソースのSQL（とヘッダー・テキスト）を以下に変更します。

```
SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, DEPTNO FROM EMP WHERE ENAME = :P2_ENAME
```

従業員名として、SQL文やSQL文の一部を入力しても、評価されるSQL文自体は変わらないことが確認できます。また、入力値は必ず従業員名として評価され、一致する従業員は存在しないため、一行も結果が返されないことが確認できます。

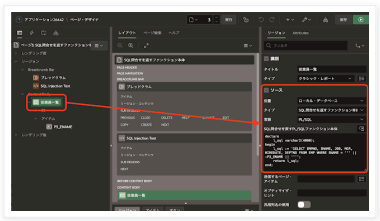
SQL問合せを返すファンクション本体ではDBMS_ASSERTを使う

ソースのタイプがSQL問合せであれば、バインド変数を使うことで、実行されるSQLが入力値によって置き換わることは起こりません。

ソースのタイプとしてSQL問合せを返すファンクション本体を選んだ場合は、SQL文はファンクション内で動的に構築します。ですので、**バインド変数を使っていてもSQLインジェクションを許してしまう**ことがあります。

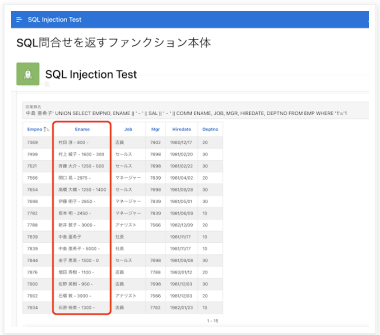
先程のページ２をコピーし、ページ３を作成します。**ソース**の**タイプ**を**SQL問合せを返すファンクション本体**に変更し、**SQL問合せを戻すPL/SQLファンクション本体**として以下のコードを設定します。

```
declare
   l_sql varchar2(4000);
begin
   l_sql := 'SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, DEPTNO FROM EMP WHERE ENAME = ''' || :P3_ENAME || '''';
   return l_sql;
end;
```



ページ・アイテムP3_ENAMEを参照する際に**バインド変数を使用していますが、置換文字列を使ったときと同じ動作になります。**

以前に使用した例である、SALとCOMMをENAMEに表示する文字列を従業員名に与えると、SAL、COMMともに表示されます。



タイプとしてSQL問合せを返すファンクション本体を選択しているので、バインド変数を使用していても、最終的にはSQL文自体にその変数の内容が含まれることになります。安全にバインド変数の内容をSQL文に含めるために、**DBMS_ASSERTパッケージに含まれるファンクションを使います。**

ページ３をページ４としてコピーし、SQLを生成するコードを以下に変更します。

```
declare
   l_sql varchar2(4000);
begin
   l_sql := 'SELECT EMPNO, ENAME, JOB, MGR, HIREDATE, DEPTNO FROM EMP WHERE ENAME = '
   || SYS.DBMS_ASSERT.ENQUOTE_LITERAL(:P4_ENAME);
   return l_sql;
end;
```

:P4_ENAMEの**クオートで囲む処理を直書きするかわりに、DBMS_ASSERT.ENQUOTE_LITERALファンクションを呼び出します。**

先程と同じ従業員名を与えると、ORA-6502が発生します。

DBMS_ASSERTパッケージの説明はこちらです。**ENQUOTE_LITERALによって、先行および後続文字を除く、すべての一重引用符が対を成していることが検証されます。**

DBMS_ASSERTの先頭にSYSをつけることにより、カレント・スキーマに同名のパッケージがあるとそちらが優先されることを回避しています。

経験上、SQLに含まれる列や表が変わる場合は動的にSQLを作成しないと対応できませんが、WHERE句については、そのまま条件を書ける場合が多いです。できれば、SQL問合せを返すファンクション本体は使わないのが望ましいです。

PL/SQL言語リファレンスも参照する

Oracle APEXでのSQLインジェクションの危険と対応方法について説明してきました。より一般的な説明が、PL/SQL言語リファレンスに含まれています。

7.4 SQLインジェクション

こちらの解説も一読をお勧めします。

Oracle APEXで作成したアプリケーションの脆弱性をレビューするツールとして、サードパーティーよりApexSecというツールが提供されています。私は使ったことがありませんが、USのOracle APEX開発部門でも使用していると伺っているので、検討に値するツールだと思います。

また、SQLにはバインド変数を必ず使うか、SQL文を動的に生成する際にはDBMS_ASSERTパッケージを呼び出せば対応できるとしても、それがアプリケーションに徹底できるかどうかは別の話になります。Virutal Private DatabaseまたはReal Application Securityを導入することにより、アプリケーションからどのようなSQLが発行されても、権限が与えられていない処理は実行しないように制限をかけることができます。RASについては以前に記事を書いていますので、そちらの方も参照していただければと思います。

以上になります。

Oracle APEXの安全なアプリケーション作成の参考になれば幸いです。

完

Yuji N. 時刻: 16:45

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示