



INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

脆弱性体験学習ツール 「AppGoat」ハンズオンセミナー

演習解説



INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

SQLインジェクションの脆弱性

[演習]

AppGoatを用いた疑似攻撃体験

IPA



- SQLインジェクションのテーマ
「不正なログイン(文字列リテラル)」
- 画面上に「Congratulations!! 」と表示されると演習クリアです。

Congratulations!!演習の目標を達成しました。

[演習解説]

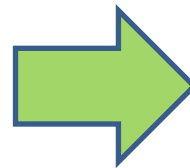
脆弱性のある箇所を特定する

- ログインID、またはパスワードにシングルクォート「'」を入力し、ログインボタンをクリックして、ウェブアプリケーションの挙動を確認しましょう。
- その結果、下記のように通常とは異なるエラーメッセージが表示されることが確認できます。

オンラインバンキング

ログインID

パスワード



✖ エラーが発生しました。

データベースエラーが発生しました。
最初のページに戻り、処理をやり直してください。

入力値により、SQL文の構文が壊れた可能性がある。
⇒SQLインジェクションの脆弱性が存在する可能性。

[演習解説]

入力値とSQL文の関係

- ログインID、パスワードに入力した値が、SQL文の下記の箇所を展開される。展開される際の処理に問題があると、SQL文の挿入が可能になる。

前提条件

ユーザ認証処理では、次のSQL文が使用されています。

```
SELECT * FROM user WHERE id = 'ログインID' AND password = 'パスワード';
```

図 5: ユーザ認証処理で使用されるSQL文

オンラインバンキング

ログインID

パスワード

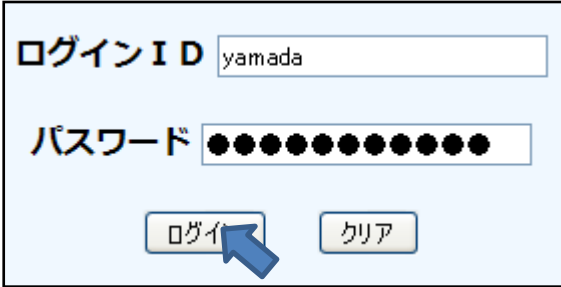
ログイン

クリア

[演習解説]

攻撃の流れを確認する

1. 攻撃者がオンラインバンキングのログインフォームのID欄に適切な文字列、パスワード欄に「' OR 'A'='A」の文字列を入力し、ログインを試みます。

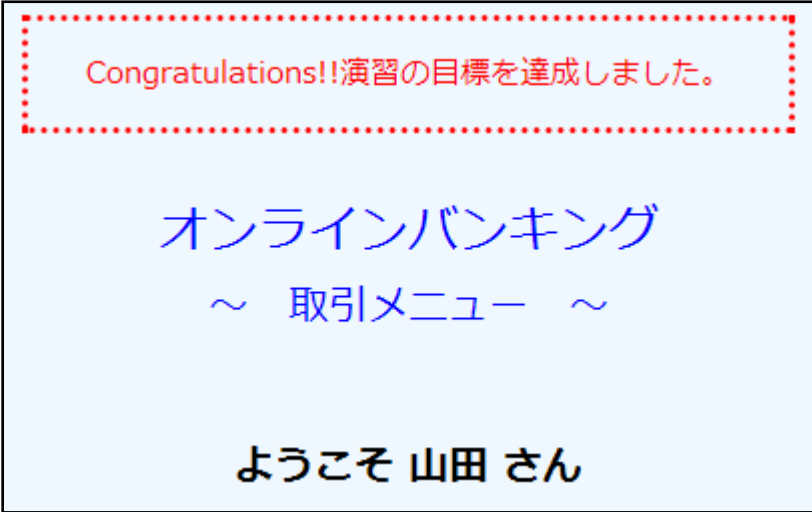


ログインID yamada

パスワード ●●●●●●●●●●●●●●●●

ログイン クリア

2. その結果、攻撃者が山田さんとしてログインできてしまいます。



Congratulations!!演習の目標を達成しました。

オンラインバンキング
～ 取引メニュー ～

ようこそ 山田 さん

[演習解説]

なぜSQL文の挿入が可能だったのか？

DBMSにおいて特別な意味を持つ記号文字「`'`」の扱いが不適切だったため。

```
SELECT * FROM user WHERE id=' $i ' AND pass=' $p ' ;
```

`$i=yamada` `$p=foo` の場合 :

```
SELECT * FROM user WHERE id=' yamada ' AND pass=' foo ' ;
```

`$i=yamada` `$p=foo' OR ' a ' = ' a` の場合 :

```
SELECT * FROM user WHERE id=' yamada ' AND pass=' foo ' OR ' a ' = ' a ' ;
```

変数中の「`'`」が文字列リテラルの区切り文字として解釈され、SQL文の構文を書き換えられてしまった。

[演習]

AppGoatを用いた疑似攻撃体験

IPA



- SQLインジェクションのテーマ
「情報漏えい(数値リテラル)」
- 画面上に「Congratulations!! 」と表示されると演習クリアです。

Congratulations!!演習の目標を達成しました。

[演習解説]

脆弱性のある箇所を特定する

- ID「yamada」でログインし、口座番号「1000001」の口座残高照会のページを参照しましょう。
- ページ参照時のURLの「account_id」パラメータ値に「a」を入力し、アクセスしてみましょう。



URL

- その結果、下記のようなエラーメッセージが表示されることが確認できます。

✖ エラーが発生しました。

データベースエラーが発生しました。
最初のページに戻り、処理をやり直してください。

入力値により、SQL文の構文が壊れた可能性がある。
⇒SQLインジェクションの脆弱性が存在する可能性。

[演習解説]

入力値とSQL文の関係

- URLのクエリストリングに入力した値の一部が、SQL文の下記の箇所を展開される。

前提条件

口座残高照会処理では、次のSQL文が使用されています。

```
SELECT * FROM account WHERE id = 'ログインID' AND account_id = 口座番号;
```

図 5: 口座残高照会処理で使用されるSQL文

http://localhost/Web/Scenario109/VulSoft/bank.php?
page=3&account_id=99 OR 1=1

[演習解説]

攻撃の流れを確認する

1. ログイン済みの攻撃者が、下記のURLにアクセスします。

```
http://localhost/Web/Scenario109/VulSoft/bank.php?page=3&
account_id=99 OR 1=1
```

2. その結果、ログイン済みの攻撃者が他人の口座残高を閲覧できてしまいます。

口座番号	残高
1000001	100,000円
1000002	200,000円
1000003	300,000円
1000004	400,000円
1000005	500,000円
1000006	600,000円

[演習解説]

なぜSQL文の挿入が可能だったのか？

変数に数値が入ることを想定している箇所に、**数値以外の文字が出力**され、文字列として扱われてしまったため。

```
SELECT * FROM account WHERE id=' $i ' AND account_id=$ai ;
```

\$i=yamada **\$ai**=99 の場合 :

```
SELECT * FROM account WHERE id=' yamada ' AND account_id=99 ;
```

\$i=yamada **\$ai**=99 OR 1=1 の場合 :

```
SELECT * FROM account WHERE id=' yamada ' AND account_id=99 OR 1=1 ;
```

変数中の「**OR**」がSQL文のOR 演算子として解釈された。



INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

クロスサイト・スクリプティングの脆弱性

[演習]

AppGoatを用いた疑似攻撃体験

IPA



- クロスサイト・スクリプティングのテーマ
「アンケートページの改ざん(反射型)」
- 画面上に「Congratulations!! 」と表示されると演習クリアです。

Congratulations!!演習の目標を達成しました。

[演習解説]

脆弱性のある箇所を特定する

- アンケートページに存在する脆弱性の箇所を探します。複数の入力欄に「'>'><s>」を入れて、アンケート内容に関するエラーページを表示してみましょう。

*1.あなたの名前を教えてください。

- エラーページ出力（HTML生成）時において、名前欄に入力した値がそのまま使われていることが確認できます。

HTMLソース

```
<div class="warning">名前に不正な文字列が含まれています。あなたの入力した名前は'>'><s>です。</div>
```

ウェブブラウザ上の表示

名前に不正な文字列が含まれています。あなたの入力した名前は'>'>です。

[演習解説]

疑似攻撃に使うスクリプトについて

- アンケートページの内容を書き換えるスクリプトを作成し、演習環境に対して、クロスサイト・スクリプティングの脆弱性を突いてみましょう。
- 例えば、アンケートページで下記のスクリプトが実行されると、アンケートページの一部が書き換わります。

```
<script>document.getElementById("account").innerHTML = '<font color="blue" size="3">もれなく一万円をプレゼントいたします。名前、住所、口座番号を入力してください。</font>';</script>
```

- HTMLのid属性値がaccountの要素の内容を、innerHTMLプロパティで置き換えています。

書き換えるHTMLの内容によって、偽の情報が表示されたり、Cookie情報を窃取されたりする。

[演習解説]

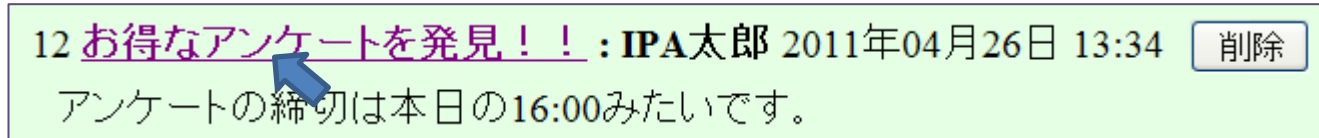
攻撃の流れを確認する



1. **攻撃者**が掲示板に罠のリンクを作成します。罠のリンクには、先ほど作成したスクリプト文字列を含めます。

*名前:	IPA太郎
*タイトル:	お得なアンケートを発見!! アンケートの締切は本日の16:00みたいです。
*本文:	
URL:	<code>http://localhost/Web/Scenario102/Vulsoft/enquete.php?page=2&sex=0&old=1&company=&xss=1&trouble=1&content=\$name<script>document.getElementById("account").innerHTML='もれなく一万円をプレゼントいたします。名前、住所、口座番号を入力してください。';</script></code>
	<input type="button" value="投稿"/> <input type="button" value="クリア"/>

2. **利用者**が罠のリンクをクリックし、アンケートページにアクセスします。



3. その結果、**利用者のウェブブラウザ上**でスクリプトが実行されます。



[演習解説]

なぜHTMLタグの挿入が可能だったのか

AppGoat

～突いてみますか？脆弱性！～

- 文字列を出力する際、「文字そのもの」として出力することを想定しているにもかかわらず、その実現に必要な処理（エスケープ処理）を実装していないため。
例：「< → <」、「> → >」
「" → "」、「& → &」など
- 「文字そのもの」として出力することを想定した箇所に「HTML タグ」として出力することができてしまうため、セキュリティ上の問題となる。

[演習]

AppGoatを用いた疑似攻撃体験

IPA



- クロスサイト・スクリプティングのテーマ
「掲示板に埋め込まれるスクリプト(格納型)」
- 画面上に「Congratulations!! 」と表示されると演習クリアです。

Congratulations!!演習の目標を達成しました。

[演習解説]

脆弱性のある箇所を特定する

- 掲示板に存在する脆弱性の箇所を探します。複数の入力欄に「'>'><hr>」を入れ投稿し、投稿結果を確認してみましょう。

*名前:	<input type="text" value="'>'><hr>"/>
*タイトル:	<input type="text" value="'>'><hr>"/>
*本文:	<input type="text" value="'>'><hr>"/>

- 投稿結果のHTMLソースを確認すると、本文欄に入力した値がHTMLの構成要素としてそのまま使われていることが確認できます。

HTMLソース

```
<div class="comment_content">'>'><hr></div>
```

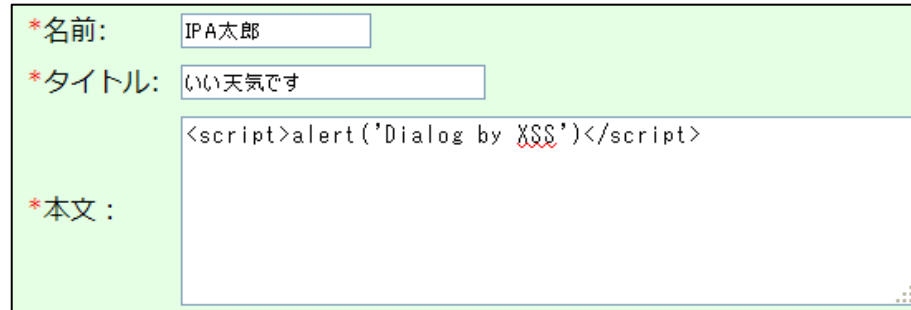
ウェブブラウザ上の表示

```
'>'>
```

[演習解説]

攻撃の流れを確認する

1. 攻撃者が掲示板にスクリプトを埋め込みます。



A screenshot of a forum post form. The form has a light green background and contains the following fields:

- *名前:
- *タイトル:
- *本文:

2. 利用者が掲示板にアクセスします。
3. その結果、利用者のウェブブラウザ上でスクリプトが実行されます。

格納型クロスサイト・スクリプティングの脆弱性は、罠サイトや罠リンクから誘導せずとも、スクリプトを実行させられてしまう。一度スクリプトが埋め込まれてしまうと、そのスクリプトを削除しない限り、当該ウェブサイトを開いただけでスクリプトが実行される。

[演習解説]

なぜHTMLタグの挿入が可能だったのか



- 学習テーマ「アンケートページの改ざん(反射型)」と同じく、HTMLにおける**エスケープ処理**を適切に実装していないため。
例：「< → <」、「> → >」
「" → "」、「& → &」など
- 「"」で括られた属性値の場合は、属性値に含まれる「"」を文字実体参照「"」にエスケープ処理する必要がある。

HTMLソース

```
<input type=text name="name" value="&quot;test&quot;">
```

ウェブブラウザ上の表示

"test"



INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

クロスサイト・リクエスト・フォージェリ (CSRF)の脆弱性

[演習]

AppGoatを用いた疑似攻撃体験

IPA



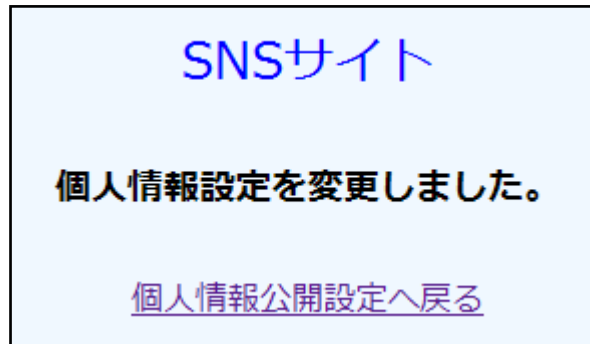
- クロスサイト・リクエスト・フォージェリのテーマ
「意図しない命令の実行」
- 画面上に「Congratulations!! 」と表示されると
演習クリアです。

Congratulations!!演習の目標を達成しました。

[演習解説]

脆弱性のある箇所を特定する

- ID「yamada」でログインし、設定変更ページのHTMLソース等からどのようなリクエストを送信しているのか確認しましょう。
- 設定変更ページから、「個人情報公開」の設定を「公開する」に変更するリクエストを送信してみましょう。



- 送信するリクエストに、第三者が予測困難な情報が含まれていないことが確認できます。

[演習解説]

攻撃の流れを確認する

1. 攻撃者が掲示板に、罨リンクを含む投稿をします。

*名前:	<input type="text" value="IPA太郎"/>
*タイトル:	<input type="text" value="ここからSNSサイトにアクセスすると"/>
*本文:	<input type="text" value="抽選で現金が当たるキャンペーン実施中"/>
URL:	<input type="text" value="http://localhost/Web/Scenario113/VulSoft/sns.php?page=4&public=1"/>
	<input type="button" value="投稿"/> <input type="button" value="クリア"/>

2. SNSサイトにログイン済みの利用者が、掲示板にある罨リンクをクリックします。

12	ここからSNSサイトにアクセスすると	: IPA太郎	2011年04月26日
18:45	<input type="button" value="削除"/>		
抽選で現金が当たるキャンペーン実施中			

3. その結果、罨リンクをクリックした利用者がSNSサイトの設定を変更するリクエストを送ってしまい、個人情報公開する設定に変更してしまいます。

[演習解説]

なぜ意図しない処理が実行されたのか？

- ログインした利用者からのみ受け付ける処理について、利用者が意図したリクエストであるかどうかを識別する仕組みがないため。

