入力処理・XSS

脆弱性の発生の場所

入力処理と脆弱性が発生する場所の関連性は次の図にようになる。

1. 原因

   HTML/表示、DB、外部コマンド、メール・・・**出力**に起因

   ファイル・・・**処理**に起因

インジェクション系の脆弱性

データの中に引用符やデリミタなどの「**データの終端**」を示す文字を**混入させ文字列の構造を変化**させる。

デリミタ(delimiter)・・・区切り文字

<例>SQLインジェクション(PHPの場合)

$sql = select * from users where id = . ‘$id’

入力された値　・・・　‘ ; delete from users —(2つのハイフン)　—はコメント

↓

select * from users where id = ’; delete from users — 2つのSQLが実行される

＃・・・コメント、行頭に付けると全体がコメントになる

入力と攻撃

攻撃手法	インタフェース	手口	データの終端
XSS	HTML	JavaScriptの注入	< “など
HTTPヘッダインジェクション	HTTP	HTTPレスポンスヘッダの注入	改行
SQLインジェクション	SQL	SQL文の注入	'など
OSコマンドインジェクション	シェルスクリプト	コマンドの注入	; | など
メールヘッダインジェクション	sendmailコマンド	メールヘッダ注入・改変	改行

入力処理

1. 入力処理の流れ

   入力(リクエスト)　→ 処理　→ 出力(レスポンス)

   文字エンコーディングの妥当性検証　・・・ 文字コードを使用した攻撃があるため、mb-check-encodingで利用して検証。

   文字エンコーディングの変換　・・・ HTTPメッセージと内部プログラムが異なる場合、mb-check-encodingで利用する。

   PHP・・・mb_convert_encodingで変換することができる。

   入力値(パラメータ文字)の妥当性検証・・・入力ミスを防ぐ、ユーザビリティ、データの不整合性を防ぐ、

2. 入力値の検証

   バイナリセーフ

   入力値がどんなバイト列であっても正しく扱うことができること。

   <例>ヌルバイトが現れても正しく処理できる

   ヌルバイト(%00)・・・**文字列の終端とみなすことが多い**、ヌルバイト攻撃に利用される。

＊%00・・・ヌルバイト(ゼロバイト)、ヌルバイトセーフの関数でない場合、文字列の終端と判断される。

<例>入力値

1**%00**<script>alert('Hello !!');</script>

%00が文字列終了と判断されて、<script>～</script>部分はチェックしない。

＊PHPのerag関数 ・・・　ヌルバイトセーフ関数ではない。現在は非推奨。

3. 入力値検証の基準

制御文字・・・タブ、改行など、通常は表示されない文字。

文字数・・・最大文字数、**SQLインジェクション**など防ぐことができる。

数値の最小、最大値・・・**DoS攻撃**を防ぐ。

<例>数値の入力時の処理

数字文字列としての文字種、文字数のチェック

文字列型から数字値型への変換

最大値、最小値の範囲にあることの確認

その他

入力項目が指定されていない、配列形式で入力されているなど。バグや脆弱性の原因なる。

4. パラメータの検証

hiddenパラメータ、ラジオボタン、select要素、cookie、セッションID、Referer、HTTPヘッダなどを行う。

表示処理に伴う問題

XSS、エラーメッセージからの情報漏洩

XSS

クッキー値が盗まれなりすましの被害にあう。サイト利用者の権限でWebアプリケーションの機能を悪用される。偽の入力フォームが表示され、個人情報を盗まれる。

＊**外部から変更できる**パラメータ・・・脆弱性があると、利用される。

1. *XSSの脆弱性*

   対策・・・メタ文字(プログラム中に特別な意味を持たせた文字のこと、<や&など)をエスケープする。

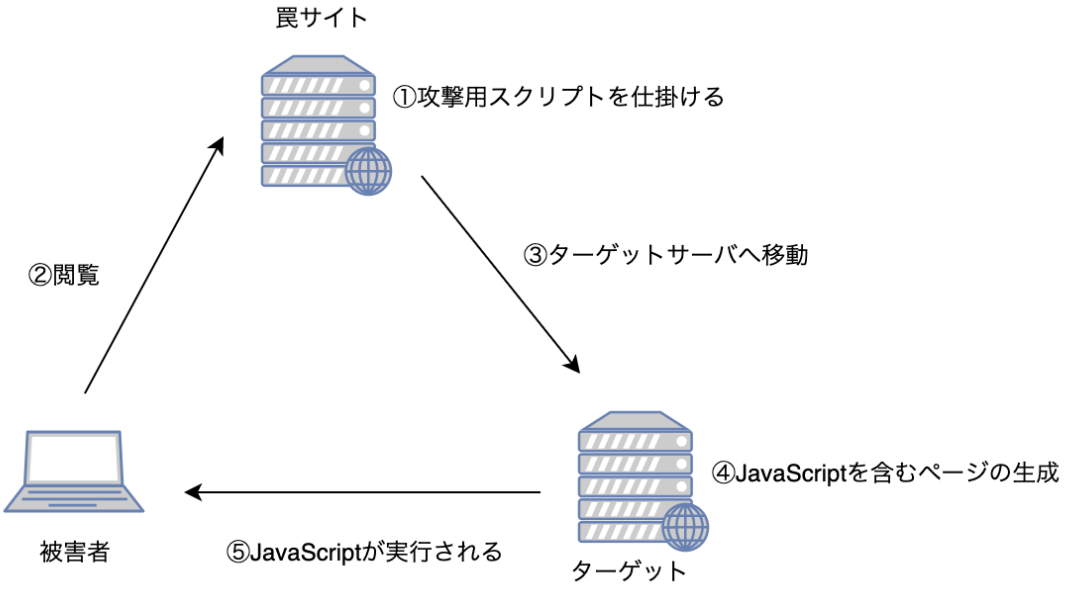
   クッキー値の盗み出し・・・セッションID (PHPSESSID)、iframeの利用、メールでセッションIDを送る。

   JavaScriptによる攻撃・・・APIは攻撃に利用される。

   画像の書き換え

   反射型XSS・・・攻撃用のJavaScriptが**攻撃サイトとは別のサイト**にある。

   接続型XSS・・・攻撃用のJavaScriptが対象の**データベースなどに保存**される。



2. 脆弱性の原因

   HTML作成時にメタ文字を適切に扱っていないことがあげられる。

   <例1>属性値を”(ダブルクォート)で囲っていない場合

   <input type=text name=name value=<?= $_GET[’$name’] ?>

   入力された値　・・・

   1+onmouseover%3dalert(document.cookie)

↓　　＊　＋・・・スペース%3d・・・＝

<input type=text name=namae value=1 onmouseover=alert(document.cookie)> に変換されてしまう。

<例2>属性値を”(ダブルクォート)で囲っている場合

<input type=”text” name=”namae” value=”<?= $_GET[‘$name’] ?>” >

入力された値　・・・” + onmouseover%3d”alert(document.cookie)

↓

<input type=”text” name=”namae” value=””onmouseover=”alert(document.cookie)”> に変換されてしまう。

XSSの対策

メタ文字をエスケープ処理する。

<例>< → <　「<」と「&」をエスケープする。

”(ダブルクォート)・・・属性値は"で囲い「<」、「”」、「&」をエスケープする。

PHPではhtmlspecialchars関数を利用する。

文字列のエンコーディングを行う。

＊XSSの保険的対策

XSSフィルタ・・・反射型XSSを検知し、無害な出力に変換する(ブラウザ機能)

X-XSS-Protectionレスポンスヘッダ・・・XSSフィルタ設定を上書きして有効にする。

<例>X-XSS-Protection:1;mode=block・・・Webサーバで設定

入力値検証(対策)

Cookie属性・・・HttpOnlyにする。　php.ini
session.cookie_httponly=on

3. href属性、src属性

URLにjavascript:～を指定するとJavaScriptが動作されてしまう。

http://example.jp/～.php?url=javascript:alert(document.cookie)

対策

URLの生成に注意

①**http、https**のみのURLを許可

http、httpsで始まるURL 、スラッシュで始まるパス(絶対パス)

②リンク先の**ドメイン名のチェック**

③リンク先のドメインが**外部ドメインの時はエラー**

④外部ドメインへのリンクは注意喚起するための**クッションページ**を表示

JavaScritpの動的生成(イベントハンドラXSS)

JavaScriptの動的生成にXSSの脆弱性が発生することがあるため、対策を行う必要がある。

JavaScriptの文字列リテラルしてエスケープすべき文字

**< > ’ ” \ → < > \’ \” \\ → < > \' \" **

escape_js関数・・・JavaScriptの文字列リテラルをエスケープする。**</script>が含まれるとソースの終端**となる。

<例></script><script>alert(document.cookie)//

JavaScriptの動的生成の対策

（１）”、’、\、改行をエスケープする。

（２）イベントハンドラの中の場合はHTMLエスケープして”で囲む。</script>という文字列が出現しないようにする。

その他

HTML、CSSタグの入力

CSS・・・expression機能でJavaScriptが動く。

<例>with:expression(document.cookie)

エラーメッセージからの漏洩

有益なアプリケーションの**内部情報が含まれる**。

エラーメッセージに秘密情報が表示させられる。