JPCERT CC®

クロスサイトリクエストフォージェリ(CSRF)とその対策

JPCERT/CC 情報流通対策グループ
脆弱性解析チーム

はじめに

- 本資料は、Web アプリケーションにおける脆弱性のひとつ、CSRF (クロスサイトリクエストフォージェリ) の仕組みとその対策に関する説明資料です。
- おもな読者層としては、Web アプリケーションを作成する開発者を想定しています。自習用の資料や勉強会での資料としてお使いください。
- 本資料によって CSRF 脆弱性に対する理解を深め、よりセキュアな Web アプリケーションの作成に役立てていただければ幸いです。

目次

- ✓ CSRF(クロスサイトリクエストフォージェリ)とは
- ✓ CSRF対策
- ✓ CSRF 脆弱性事例
- ✓ CSRF対策ライブラリの紹介
- ✓ まとめ
- ✓ 参考情報

CSRF(クロスサイトリクエストフォージェリ)とは

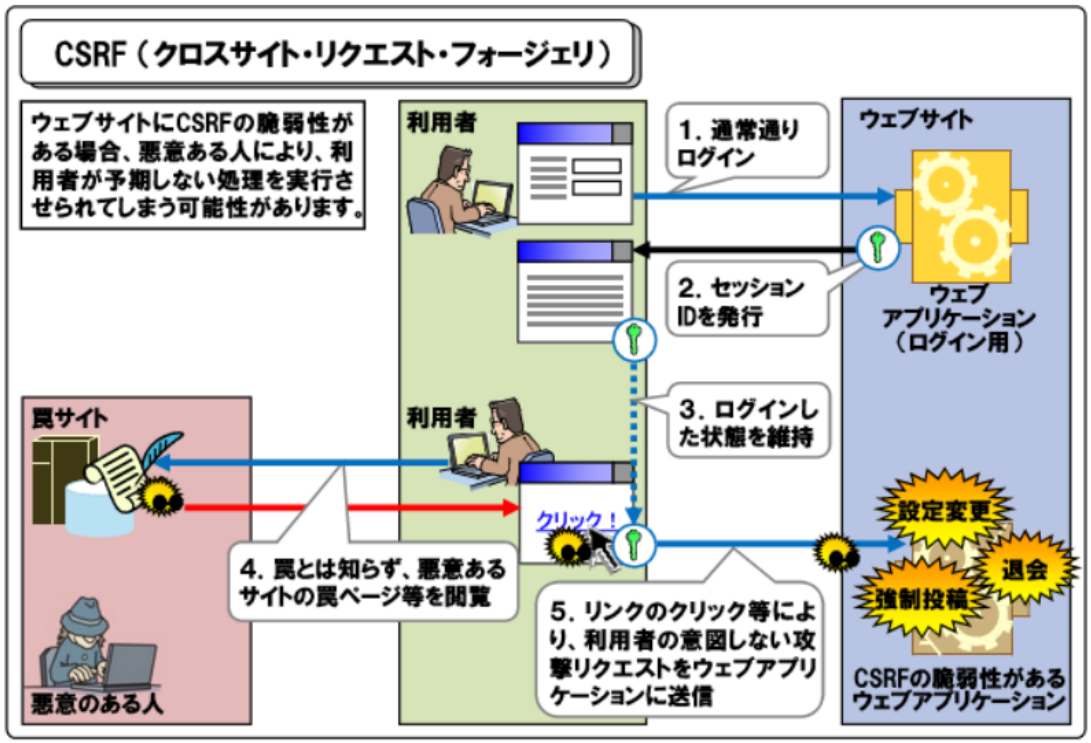
CSRF(クロスサイトリクエストフォージェリ) とは

- **CSRF (Cross-Site Request Forgery, クロスサイトリクエストフォージェリ)**とは、Web アプリケーションに対する攻撃手法の一種です。また、CSRF が可能となるような脆弱性、ということでそのまま脆弱性の名称としても使われ、「Web アプリケーション A には CSRF の脆弱性が存在する」というような言い方をします。

Web アプリケーションの脆弱性としては**CSRF** の他にも例えば、**XSS (クロスサイトスクリプティング)**、**SQL インジェクション**、**ディレクトリトラバーサル**など、様々なものが知られています。

IPA「安全なウェブサイトの作り方」では…

ログインした利用者からのリクエストについて、その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を持たないウェブサイトは、外部サイトを経由した悪意のあるリクエストを受け入れてしまう場合があります。このようなウェブサイトにログインした利用者は、悪意のある人が用意した罠により、利用者が予期しない処理を実行させられてしまう可能性があります。



独立行政法人 情報処理推進機構(IPA)
「安全なウェブサイトの作り方」
1.6 CSRF より

CSRF攻撃の特徴

- ログイン認証などで保護されているWebアプリケーションでも、**正規ユーザがログインした状態で罠ページをアクセスすることで攻撃が成立する**

- Webアプリケーションに**直接アクセスするのは**、罠ページから誘導された**正規ユーザ**

CSRF脆弱性が悪用された場合のリスク

意図していない機能を実行させられる

- 例えば、掲示板サイトでは、攻撃者によって意図しない投稿をさせられる可能性
- オンラインショッピングサイトでは、攻撃者によって意図しない商品購入をさせられる可能性

CSRF攻撃の流れを詳しく見る

CSRF対策を考えるために、CSRF攻撃の例として掲示板への書き込みをさせられる場合の処理の流れを詳しく見ておきましょう。

まずユーザが意図して書き込みを行う場合、つまり通常の入力フォームの処理を図解します。次に、CSRF攻撃が行われる場合を図解します。

通常の入力フォームの処理(0)

掲示板サイトに対する操作は2回のアクセスを通じて行われます。

- 1回めのアクセス: 入力フォームを取得
- 2回めのアクセス: 入力フォームにしたがってリクエストを送信

1. ユーザは掲示板にログインし、書き込みのためのページにアクセス
2. 書き込み用の入力フォームを含むページが表示される
3. フォームへの入力を行い、書き込みリクエストを送信
4. サーバは書き込みリクエストを受け付け、その処理結果を返す

通常の入力フォームの処理(1)

HTTP REQUEST

GET form.html

タイトル

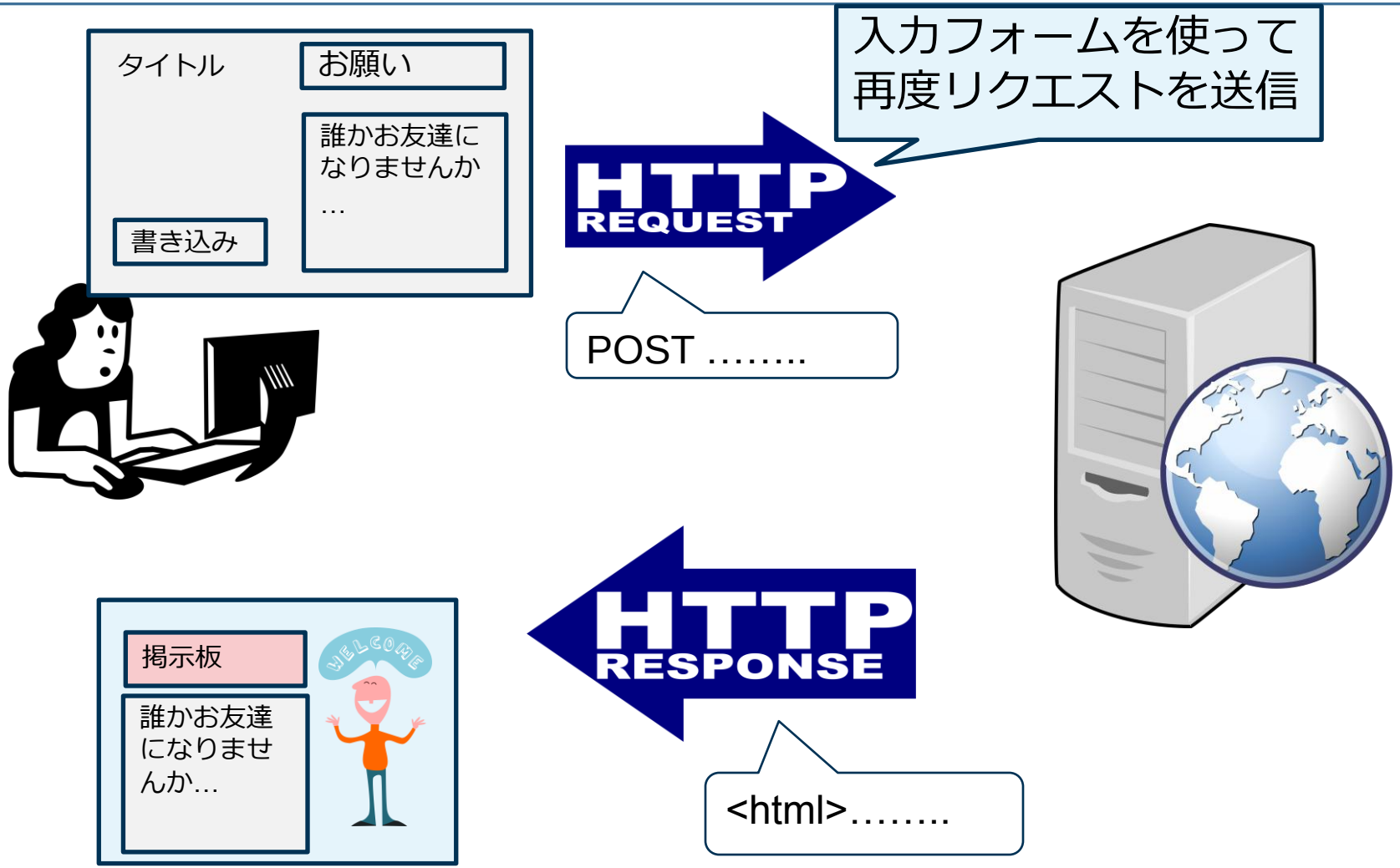
書き込み

HTTP RESPONSE

<form>……..

サーバから送られてきた入力フォームを表示

通常の入力フォームの処理(2)



CSRF攻撃(0)

CSRF攻撃は罠サイトへのアクセスから始まります。

- 1回めのアクセス: **罠サイトから**入力フォームを取得
- 2回めのアクセス: 入力フォームにしたがってリクエストを送信

1. ユーザは掲示板サイトにログインした状態で、**罠ページにアクセス**
2. 一件無害な内容に偽装したページが表示される
3. 偽装されたボタンをクリックすると、裏に仕込まれていた書き込みリクエストを掲示板サイトに送信
4. サーバは書き込みリクエストを受け付け、その処理結果を返す

※この図解では、罠ページにアクセスした後、ボタンをクリックすることでリクエストが送信されますが、ページにアクセスしただけでリクエストを送信させるように罠ページをつくることも可能です。

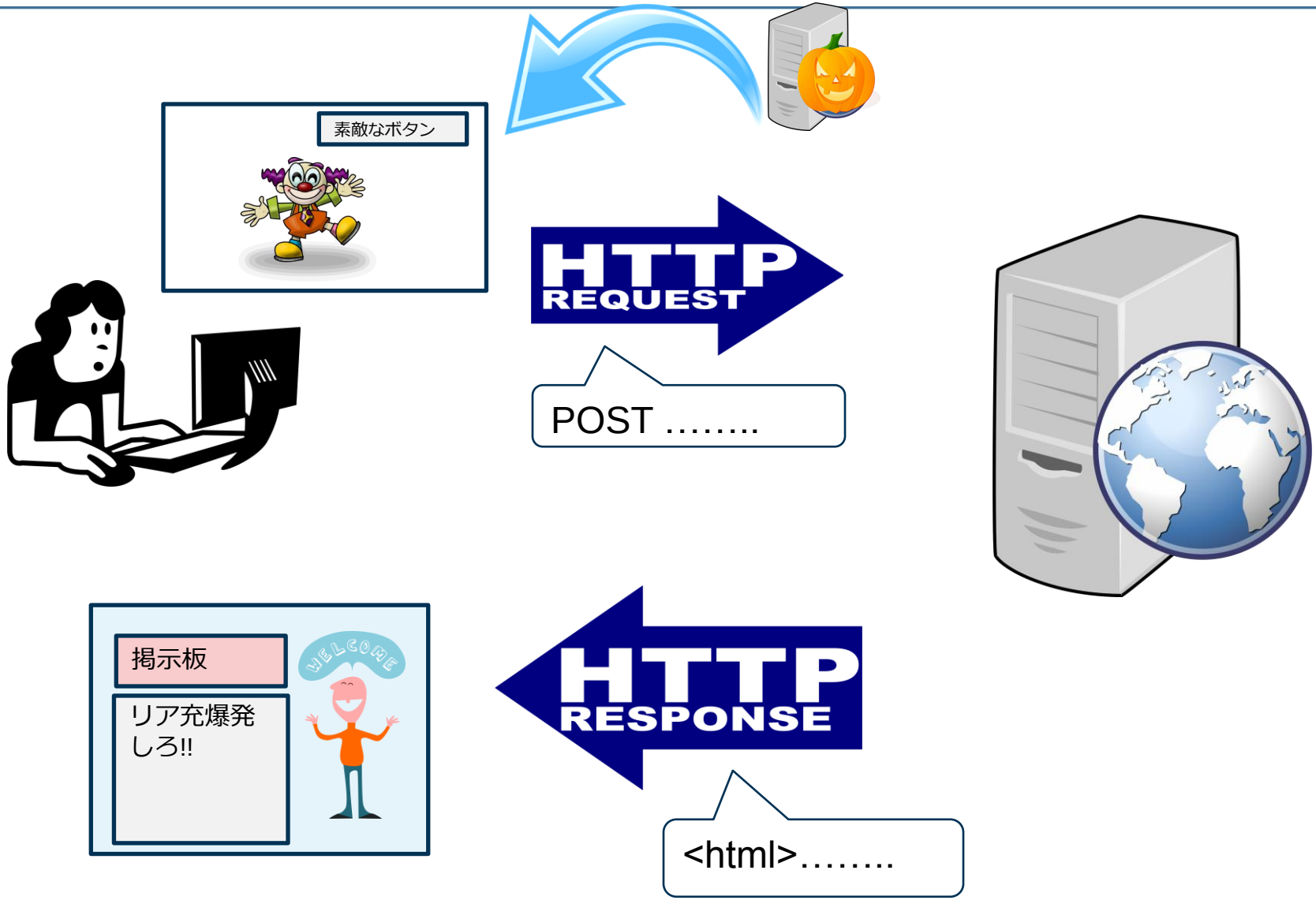
CSRF攻撃(1)

GET xxxx.html

<form>……..

サーバから送られてきたページを表示

CSRF攻撃(2)



通常の処理とCSRF攻撃の違い

- Webブラウザから見ると…
  - —入力フォームがどこから来るかが違う
  - —正規サイトから来るか、罠サイトから来るか
- Webアプリケーションから見ると…
  - —どちらの場合も正規ユーザからのリクエスト

どうやって対策する?

CSRF(クロスサイトリクエストフォージェリ)対策

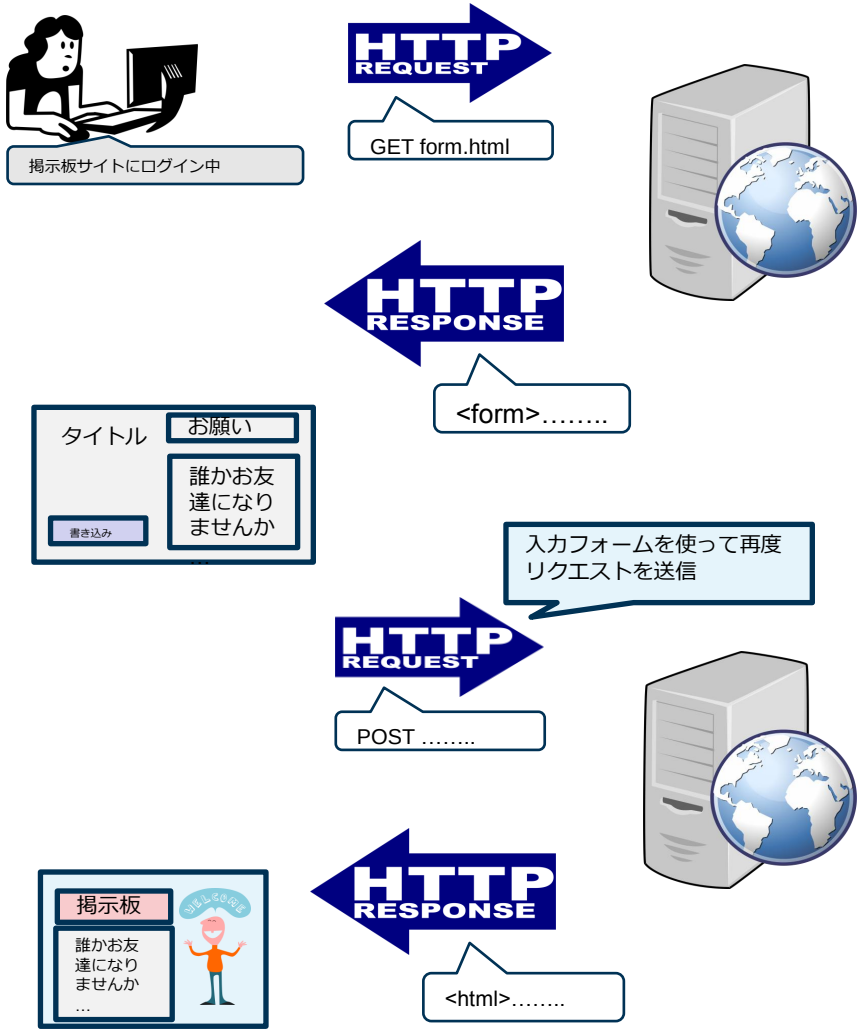
どうやってCSRF対策する?

- Webアプリケーション側で対策したい
  - しかしCSRF攻撃でもリクエストを送るのは正規ユーザ
  - 単純に送信者を見るだけでは区別できない
- ヒント
  - Webアプリケーションに対する操作は2回のアクセスによって行われている

  • 1回めのアクセス: 入力フォームを取得
  • 2回めのアクセス: 入力フォームにしたがってリクエストを送信

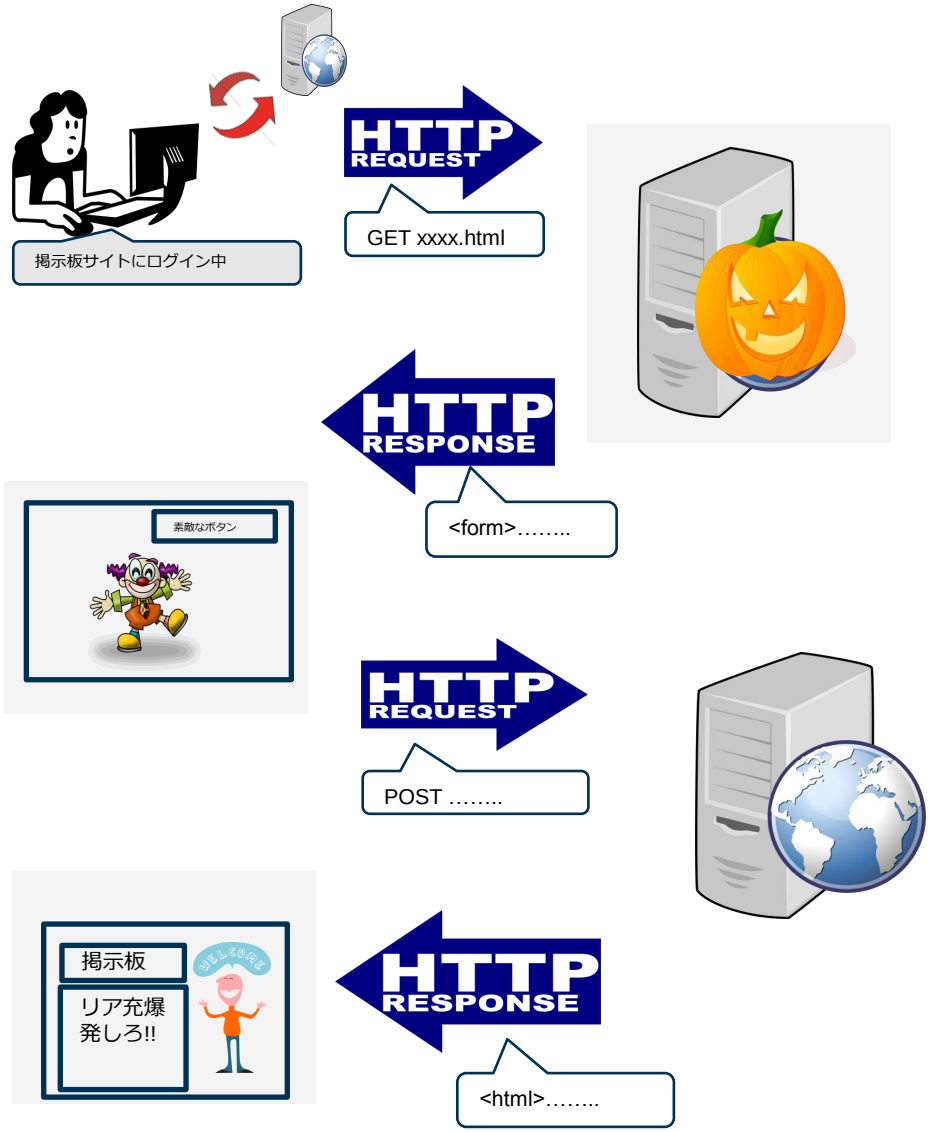
もういちど流れを確認してみよう 

通常の処理



CSRF攻撃を受ける場合

CSRF攻撃で使われるリクエストは、攻撃者が勝手につくったもの.



CSRF対策のアイディア

CSRF攻撃で使われるリクエストは、攻撃者が勝手につくったもの.

正規のリクエストと偽装されたリクエストを識別できれば、サーバ側で対処できる!

最初にブラウザに返すフォームのなかに、**攻撃者が推測できない情報**を入れておけばよい!!

nonce を使ったCSRF対策

- 1回めのアクセスでWebブラウザに返す入力フォームに**外部からは推測できない情報**をあらかじめ埋め込んでおく。
  - —この目的のために**乱数(nonce)**を使うことが多い
- 2回めのアクセスで送られてきたリクエストを処理する際nonceの**存在**と値の**一致**を確認することで、正規のリクエストか否かを識別する。

参考: IPA「セキュアプログラミング講座、Webアプリケーション編」第4章セッション対策

nonce を使ったCSRF対策

- Webアプリケーションフレームワークの多くはCSRF対策に関する機能を提供している。
  - “Most frameworks have built-in CSRF support such as Joomla, Spring, Struts, Ruby on Rails, .NET and others.”

    https://www.owasp.org/index.php/CSRF
  - IPA「セキュアプログラミング講座」では Ruby-on-Rails の例を紹介(2007年時点の資料)

CSRF脆弱性事例

事例1: Spacewalk

製品名：Spacewalk

- 製品概要： Red Hat が提供するLinuxシステムの統合管理ツールRed Hat Network SatelliteのOSS版
- 製品URL：http://spacewalk.redhat.com/

- CVE：CVE-2009-4139
- アドバイザリURL：
  - http://www.redhat.com/support/errata/RHSA-2011-0879.html
  - http://securitytracker.com/id?1025674
  - https://bugzilla.redhat.com/show_bug.cgi?id=529483
  - http://xforce.iss.net/xforce/xfdb/68074
- 脆弱性指摘バージョン：1.2.39
- 脆弱性修正バージョン：1.2.39-85

脆弱なコードのポイント

```
<form method="POST" action="/rhn/users/DeleteUserSubmit.do?uid=2">
<div align="right">
      <hr />
      <input type="submit" value="ユーザーの削除" />
    </div>
</form>
```

Nonceが存在せず、入力フォームの偽装が可能!!

- 例えば、CSRF攻撃によりユーザー削除が可能。

SpacewalkにおけるCSRFの修正

- カスタムタグを定義し、リクエストの内容にCSRF対策トークンを追加している。
  - SpacewalkではWebフレームワーク Apache Struts を利用しており、カスタムタグは Apache Struts の機能
  - フォームが含まれるJSPファイル(348個)へカスタムタグを挿入

```
<form method="POST"
      action="/rhn/users/DeleteUserSubmit.do?uid=${param.uid}">
    <rhn:csrf />
    <html:submit styleClass="btn btn-danger">
        <bean:message key="deleteuser.jsp.delete"/>
    </html:submit>
</form>
```

この修正により、Webブラウザに渡される入力フォームにnonceが設定され、入力フォームの偽装が困難になった.

事例2: Login Rebuilder

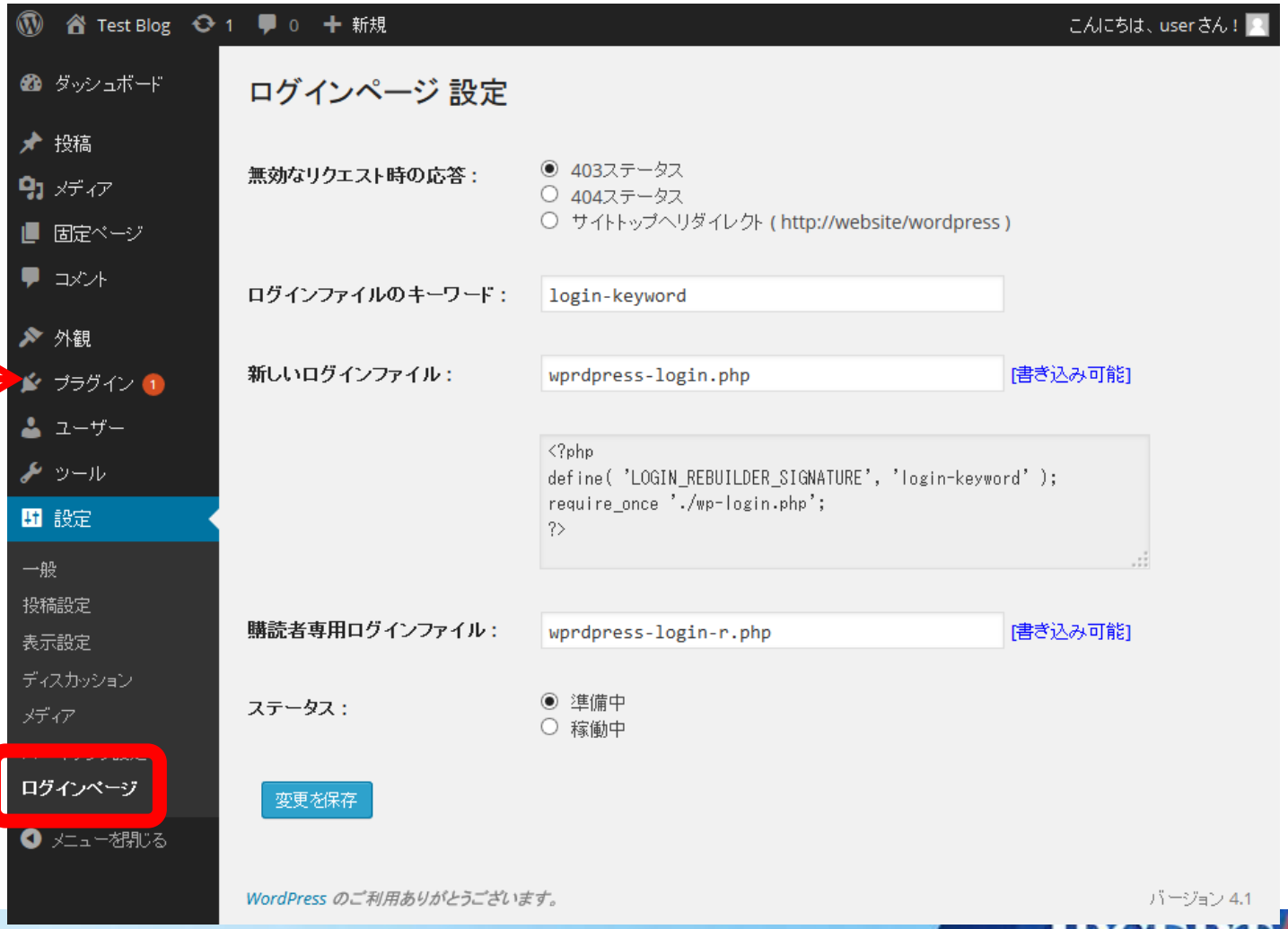
製品名：Login Rebuilder

- 製品概要：ログインページをデフォルトのwp-login.phpから変更する事でセキュリティを向上するWordPressプラグイン。
- 製品URL：
  - https://wordpress.org/plugins/login-rebuilder/
  - http://plugins.svn.wordpress.org/login-rebuilder
- CVE：CVE-2014-3882
- アドバイザリURL：
  - http://wordpress.org/plugins/login-rebuilder/changelog/
  - http://12net.jp/news/n20140623_01.html
  - https://jvn.jp/jp/JVN05329568/index.html
- 脆弱性指摘バージョン：1.1.3(リビジョン868421)
- 脆弱性修正バージョン：1.2.0(リビジョン914619)

脆弱なフォームの場所

- WordPress管理画面のLogin Rebuilderの設定フォーム(「設定」→「ログインページ」)
- http://website/wordpress/wp-admin/options-general.php?page=login-rebuilder-properties

脆弱なバージョンは最新ではないため、更新があることを知らせる赤い数字が表示されている。



formタグ内の内容

nonceが存在せず、入力フォームの偽装が可能!!

```
<form method="post" action="/wordpress/wp-admin/options-general.php?page=login-rebuilder-
properties">
(省略)
<input type="radio" name="properties[response]" id="properties_response_1"
value="1" checked='checked' />
<input type="radio" name="properties[response]" id="properties_response_2" value="2" />
<input type="radio" name="properties[response]" id="properties_response_3" value="3" />
(省略)
<input type="text" name="properties[keyword]" id="properties_keyword" value="login-keyword"
class="regular-text code" />
(省略)
<input type="text" name="properties[page]" id="properties_page" value="wprdpress-login.php"
class="regular-text code" />
(省略)
<textarea  name="properties[content]" id="login_page_content" rows="4" cols="60" style="font-
family:monospace;" readonly="readonly"></textarea>
(省略)
<input type="text" name="properties[page_subscriber]" id="properties_page_subscriber"
value="wprdpress-login-r.php" class="regular-text code" />
(省略)
<input type="radio" name="properties[status]" id="properties_status_0" value="0"
checked='checked' />
<input type="radio" name="properties[status]" id="properties_status_1" value="1"  />
(省略)
<input type="submit" name="submit" value="変更を保存" class="button-primary" />
</form>
```

Login RebuilderのCSRF修正バージョンのポイント

- WordPressのwp_create_nonce()を使用して一時的な値 $nonce を生成し、フォームの既存のキー(properties[response])に追加。

- nonce生成部分

```
$nonce = wp_create_nonce(
self::LOGIN_REBUILDER_PROPERTIES_NAME.'@'.$wp_version.'@'.LOGIN_REBUILDER_DB_VERSION);
```

- input要素のname属性の値への追加例 (properties[page]の場合、他も同様)

```
<input type="text" name="properties_<?php echo $nonce; ?>[page]" id="properties_page"
value="<?php _e( $this->properties['page'] ); ?>" class="regular-text code" />
```

- nonceの検証部分(POST内容のキー名の存在確認)

```
if ( isset( $_POST['properties_'.$nonce] ) ) {
```

Login RebuilderのCSRF修正バージョンのポイント

- 修正後のHTMLのinput要素 (properties[page]の場合)

```
<input type="text" name="properties_c6808428a6[page]" id="properties_page"
value="wprdpress-login.php" class="regular-text code" />
```

- inputのname属性を「properties[response]」から「properties_**c6808428a6**[response]」といった一時的なものを使用するように変更
- フォームからPOSTされた内容のキーがproperties_**c6808428a6**ではない場合は変更を行わない

Login Rebuilderのnonceを用いたCSRF対策の実装は、独立したinput要素ではなく既存のinput要素のname属性の値を変更するという形で行われている。
(今回確認したバージョン1.2.0の場合)

参考: WordPress本体が提供するnonce関連の関数(1)

- WordPressはnonceを取り扱う関数群が用意されており、プラグインやテンプレートで使用できる。
  - WordPressのnonceに関するページ：https://codex.wordpress.org/WordPress_Nonces
  - 関数リファレンス： http://wpdocs.osdn.jp/関数リファレンス

- nonce を取り扱う関数は関数名に nonce という文字列が含まれる

参考: WordPress本体が提供するnonce関連の関数(2)

使用例(関数リファレンスより)：

- nonceの生成(wp_create_nonceを使用)：リンク先URLにnonceを挿入している

```
<?php $nonce= wp_create_nonce  ('my-nonce'); >
<a href='myplugin.php?_wpnonce=<?php echo $nonce ?>'>
```

- nonceの確認(wp_verify_nonceを使用)：検証に失敗したら終了(die)する

```
<?php
    $nonce=$_REQUEST['_wpnonce'];
    if (! wp_verify_nonce($nonce, 'my-nonce') )
      die('Security check');
?>
```

※ この他、プラグインの設定ページなどのような管理ページにおける nonce の検証処理を想定した check_admin_referer 関数や check_ajax_referer 関数が提供されている。

CSRF脆弱性事例を探す

さらに実際の事例を調べるには、脆弱性情報を集めたサイトで検索すると便利

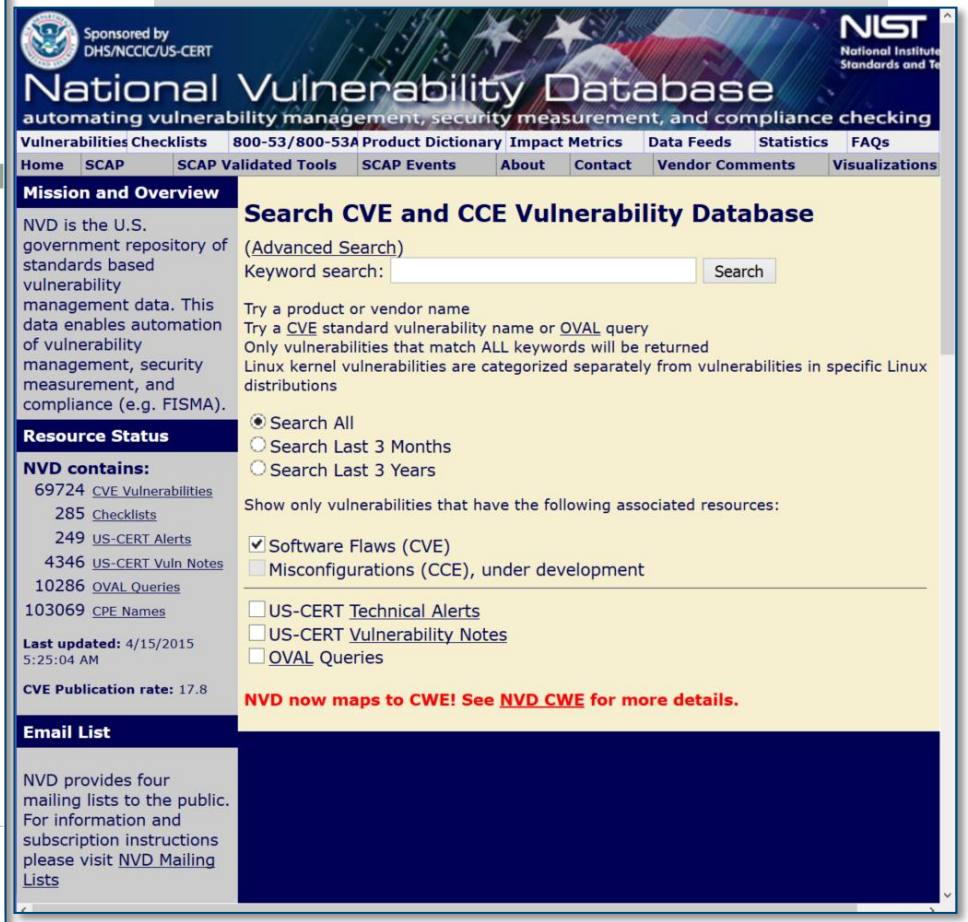
- CVE (https://cve.mitre.org/)
- NVD (https://nvd.nist.gov/)
- JVN (https://jvn.jp/)
- OSVDB (http://www.osvdb.org/)

CSRF脆弱性事例を探す: CVE と NVD

https://cve.mitre.org/cve/cve.html

https://nvd.nist.gov/

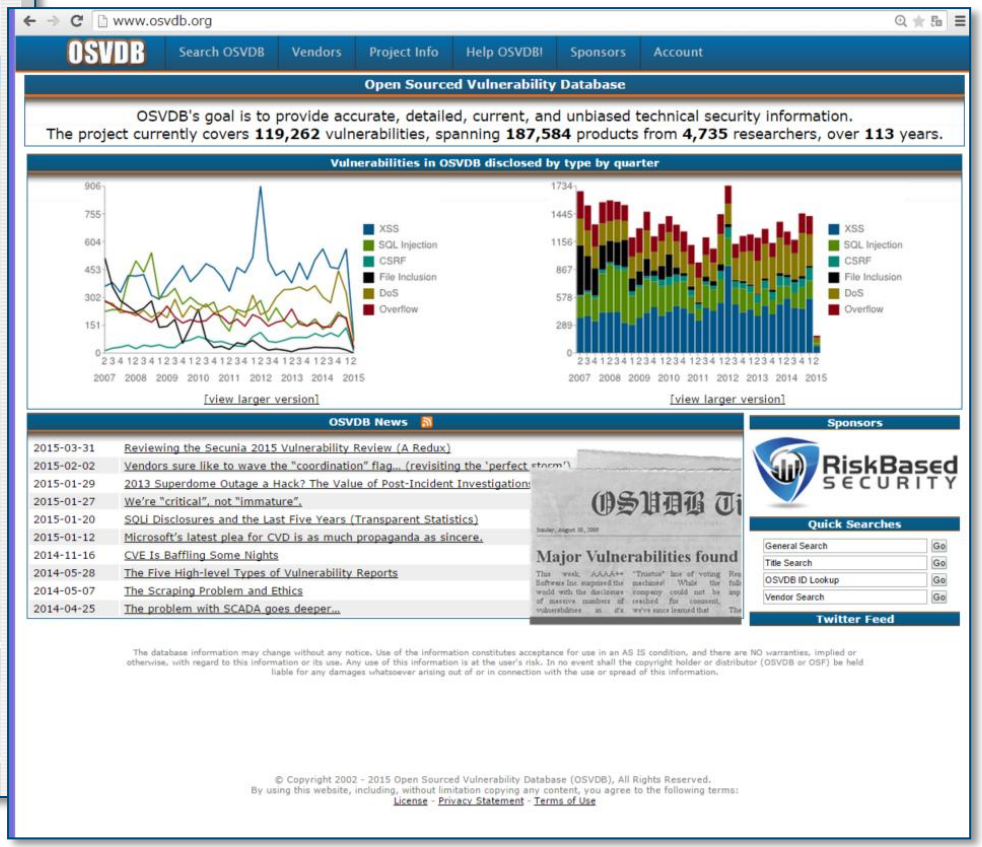
https://web.nvd.nist.gov/view/vuln/search



CSRF脆弱性事例を探す: JVN と OSVDB

https://jvn.jp/

http://www.osvdb.org/



CVEでCSRF脆弱性事例を探してみると……

- 2001年から2014年の範囲でCVEの概要(Description)に「CSRF」を含むもの、で検索すると1039件存在
  - CVEでは対象製品で使われているプログラミング言語の情報は記載されていないため、どの言語で実装されたソフトウェアにCSRFが多く発見されているかは分からない
  - CVEの概要の中に問題となるファイル名 foo.phpやBarServletなどが記載され、PHPやJavaで記載されていると推測できる場合もある
- JVNの脆弱性レポートでは39件存在
  - PHP: 13件、Perl: 2件、Ruby: 1件など

CSRF脆弱性事例 (JVN 掲載案件から抜粋)

JVN番号	タイトル	CVE番号
JVN#32631078	複数の ASUS 製無線 LAN ルータにおけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-7270
JVN#94409737	WordPress 用プラグイン MailPoet Newsletters におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-3907
JVN#42511610	acmailer におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-3896
JVN#36259412	Web給金帳におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-3881
JVN#05329568	WordPress 用プラグイン Login rebuilder におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-3882
JVN#13313061	東芝テック製 e-Studio シリーズにおけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-1990
JVN#50943964	phpMyFAQ におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2014-0813
JVN#11221613	EC-CUBE におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2013-5993
JVN#48108258	HP ProCurve 1700 シリーズのスイッチにおけるクロスサイトリクエストフォージェリの脆弱性	CVE-2012-5216
JVN#06251813	複数のサイボウズ製品におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2013-2305
JVN#59503133	複数の NEC 製モバイルルータにおけるクロスサイトリクエストフォージェリの脆弱性	CVE-2013-0717
JVN#53269985	Welcart におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2012-5178
JVN#44913777	せん茶SNS におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2012-1237
JVN#83459967	Janetter におけるクロスサイトリクエストフォージェリの脆弱性	CVE-2012-1236

CSRF(クロスサイトリクエストフォージェリ)対策ライブラリの紹介

CSRF対策ライブラリとは

フォームへのnonce埋め込み、受信したフォームのnonceチェックなどCSRF対策のための処理を支援するライブラリ

- OWASP CSRFGuard などがそういうものらしい
- どういう仕組みなの?
- Webアプリケーションフレームワークで提供されているものと何か違うの?

ここでは、CSRF対策ライブラリがどういうものか簡単な説明と、すでに紹介したCSRF脆弱性事例に独自にCSRF対策ライブラリ適用を試みた様子を紹介します。
CSRF対策を検討する際の参考にしてください。

CSRF対策ライブラリとは

Java, PHP, PythonにおけるCSRF脆弱性対策ライブラリの例.

言語	名称	概要
Java	Apache Tomcat CSRF Prevention Filter	Apache Tomcatのサーブレットコンテナが提供するFilter群のひとつ
Java	OWASP CSRFGuard	サーブレットWebアプリケーションに組み込むフィルタ
Java	Spring Security	Spring Framework上で使用できるServletFilter群
Java	csrf-filter	サーブレットWebアプリケーションに組み込むフィルタ
PHP	OWASP CSRFProtector	PHPの他、Apacheのモジュールとしても提供される
PHP	csrf-magic	PHPに読み込むだけで使用可能なライブラリ
Python	CSRF Protection middleware	PythonのWebアプリフレームワークの CSRF対策機能

※Web フレームワークなどで提供されている機能も含んでいます

- TIOBEによるプログラミング言語ランキングにおいてwebアプリケーション開発に使用される言語の中ではJava, PHP, Pythonが上位に位置している.
- http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
- (2015年2月の時点ではC, Java, C++, Objective-C, C#, JavaScript, PHP, Pythonと続く)

CSRF対策ライブラリの実装アプローチ

- Java の場合
  - アプリケーションサーバ(tomcatなど)のフィルタ機能として実装
  - Webアプリケーションフレームワークの機能を利用してWebアプリケーションの手前に実装

- PHPの場合
  - PHPの**ob_start()**による出力バッファに対する処理などを使って実装

各対策ライブラリの概要


Apache Tomcat CSRF Prevention Filter


OWASP CSRFGuard


Spring Security


csrf-filter


OWASP CSRFProtector


csrf-magic


Django CSRF Protection middleware

各対策ライブラリの概要

Apache Tomcat CSRF Prevention Filter

OWASP CSRFGuard

Spring Security

csrf-filter

OWASP CSRFProtector

csrf-magic

Django CSRF Protection middleware

CSRF Prevention Filter概要

- 名称：CSRF Prevention Filter
  - プロジェクトURL： http://tomcat.apache.org/
  - ドキュメントURL： http://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#CSRF_Prevention_Filter
- 概要：
  - Apache Tomcatのサーブレットコンテナが提供するFilter群のひとつ
  - CSRF対策としてnonceを用いた検証を行う
  - ミドルウェアであるTomcat内にCSRF対策ライブラリに相当するクラスがある為、本ライブラリを適用したWebアプリケーションは、Jetty等の他のサーブレットコンテナではクラスが参照できず動作しない
- 対策原理：
  - JavaサーブレットのFilter機構を用いてクライアントとサーブレット本体の間に割り込み、リクエストにnonce付加、レスポンス中のnonce検証を行う
- 適用手順：
  1. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する
  2. クライアントへ応答するWebページを生成する JSP の form要素の部分を書き換える

CSRF Prevention Filter適用方法(1)

1. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する

```
<filter>
  <!-- CSRF Prevention Filterの宣言 -->
  <filter-name>CsrfFilter</filter-name>
  <filter-class>org.apache.catalina.filters.CsrfPreventionFilter
  </filter-class>
  <!-- パラメータの設定例。詳細は次ページに記載 -->
  <init-param>
    <param-name>entryPoints</param-name>
    <param-value>/entry.jsp</param-value>
  </init-param>
</filter>

<!-- 宣言したCSRF Prevention Filterの適用範囲の設定 -->
<filter-mapping>
    <filter-name>CsrfFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

CSRF Prevention Filter適用方法(2)

init-paramの項にて指定できるパラメータ

param-name	説明	デフォルト値
denyStatus	リクエスト拒否時のHTTPレスポンスコード。	403
entryPoints	nonceのチェックを行わないページをカンマ区切りで記載。 ここにWebサイトの入口や処理終了時のページを指定する。	—
nonceCacheSize	並列の要求を処理できるようにするためにキャッシュするnonceの数	5
randomClass	nonce生成に用いるクラス。 java.util.Randomのインスタンスの必要あり。	java.security.SecureRandom

CSRF Prevention Filter適用方法(3)

2. クライアントへ応答するWebページを生成する JSP の form 要素の部分を書き換える

- 変更前：

```
<form action="path/to/servlet" method="POST">
```

- 変更後：

```
<%
  String urlAction = "path/to/servlet";
  String urlActionEncoded = response.encodeURL(urlAction);
%>
<form action="<%=urlActionEncoded %>" method="POST">
```

CSRF Prevention Filter適用方法(4)

- CSRFから保護したいformの送信先である ”path/to/servlet” を HttpServletResponse#encodeURL(String) メソッドで処理する

- encodeURLメソッドはCSRF対策固有のものではなく、クッキー利用不可時にセッションIDを付与するもの
  (https://tomcat.apache.org/tomcat-7.0-doc/servletapi/javax/servlet/http/HttpServletResponse.html#encodeURL(java.lang.String))

- 実行時には以下のようにnonceがformの送信先 path/to/servlet の後ろに付与される

```
<form
action="path/to/servlet;jsessionid=7F7A056EC7C094FA2A372318525EE5CB?org.apache.catalina.filters.CSRF_NONCE=521334A9F01F17C4B0D3D847A221A9BB" method="POST">
```

各対策ライブラリの概要

Apache Tomcat CSRF Prevention Filter


OWASP CSRFGuard

Spring Security

csrf-filter

OWASP CSRFProtector

csrf-magic

Django CSRF Protection middleware

OWASP CSRFGuard概要

- 名称： OWASP CSRFGuard
  - プロジェクトURL： https://www.owasp.org/index.php/CSRFGuard
  - ドキュメントURL： https://www.owasp.org/index.php/CSRFGuard_3_User_Manual
- 概要：
  - OWASPが提供するサーブレット向けのCSRF対策ライブラリ
  - CSRF対策としてnonceを用いた検証を行う
  - フォームへのnonceの埋め込みにJavaScriptを使用しているため、JavaScriptが使用できない環境では動作しない
  - 独立したjarファイルとして配布されているため、サーブレットコンテナの種類に依存しない
- 対策原理：
  - JavaサーブレットのFilter機構を用いてクライアントとサーブレット本体の間に割り込み、リクエストにnonce付加、レスポンス中のnonce検証を行う
- 適用手順：
  1. アプリケーションの classpath へ 本ライブラリの jar を追加する
  2. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する
  3. 設定ファイル Owasp.CsrfGuard.properties に設定を記載する
  4. クライアントへ応答するWebページを生成する JSP を書き換える

OWASP CSRFGuard適用方法(1)

1. アプリケーションの classpath へ Owasp.CsrfGuard.jar を追加
   - 例えばWebアプリケーションの /WEB-INF/lib ディレクトリ内に配置する

OWASP CSRFGuard適用方法(2)

2. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する

```
<!- OWASP CSRFGuardのListenerの宣言 -->
<listener><listener-class>org.owasp.csrfguard.CsrfGuardServletContextListener</listener-class></listener>
<listener><listener-class>org.owasp.csrfguard.CsrfGuardHttpSessionListener</listener-class></listener>
<!- 設定ファイルのパスの指定 次ページで説明 -->
<context-param>
    <param-name>Owasp.CsrfGuard.Config</param-name>
    <param-value>WEB-INF/Owasp.CsrfGuard.properties</param-value>
</context-param>
<!- OWASP CSRFGuardのFilterの宣言 -->
<filter>
    <filter-name>CSRFGuard</filter-name>
    <filter-class>org.owasp.csrfguard.CsrfGuardFilter</filter-class>
</filter>
<!-- 宣言したCSRF Prevention Filterの適用範囲の設定 -->
<filter-mapping>
    <filter-name>CSRFGuard</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- ページにトークンを挿入するJavaScriptのServletの宣言 -->
<servlet>
     <servlet-name>JavaScriptServlet</servlet-name>
     <servlet-class>org.owasp.csrfguard.servlet.JavaScriptServlet</servlet-class>
</servlet>
<!-- JavaScriptのServletのパス指定 -->
<servlet-mapping>
     <servlet-name>JavaScriptServlet</servlet-name>
     <url-pattern>/JavaScriptServlet</url-pattern>
</servlet-mapping>
```

OWASP CSRFGuard適用方法(3)

3. WEB-INF/Owasp.CsrfGuard.properties に設定を記載する。

指定できる主な項目

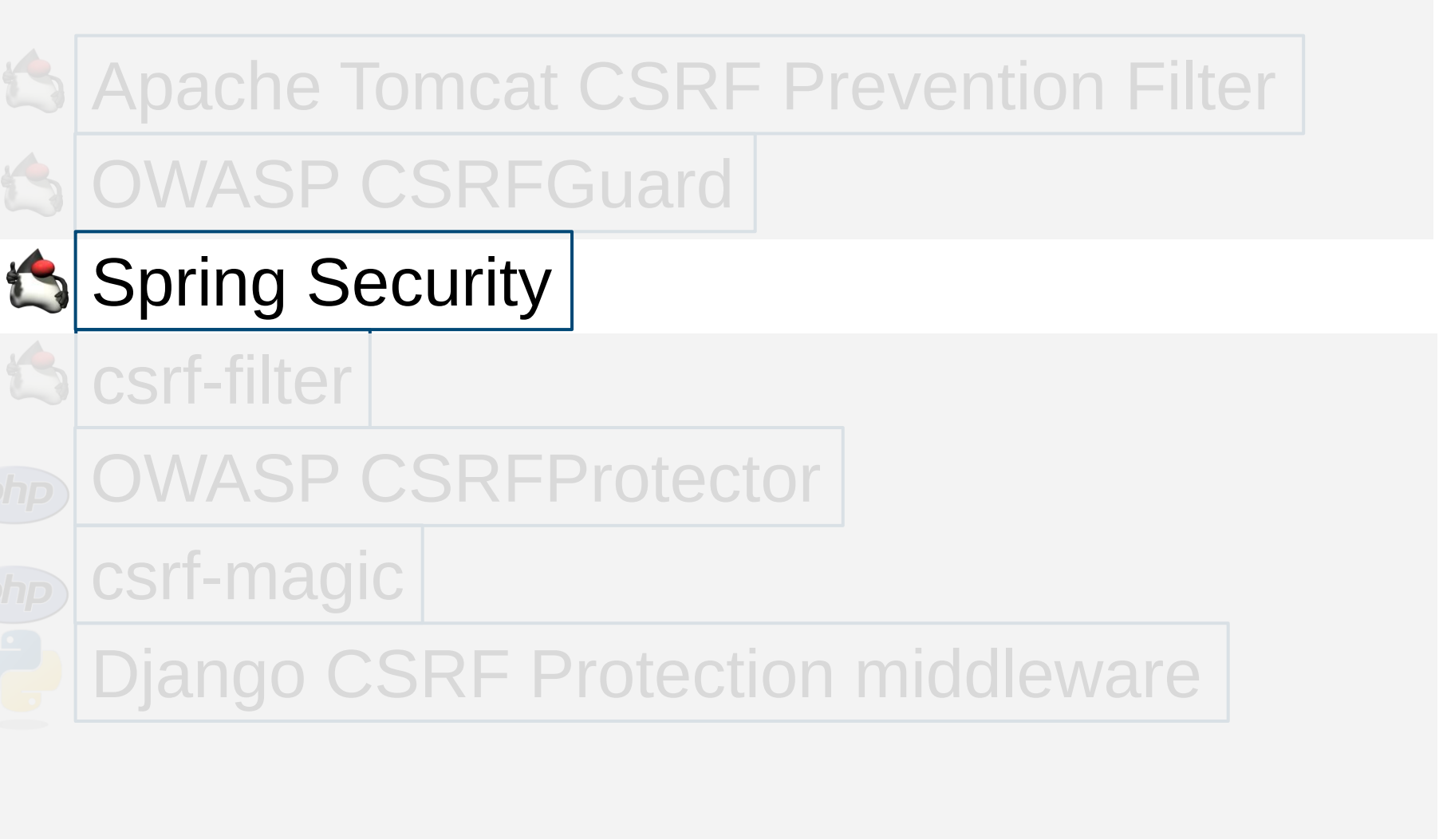
設定キー	説明
org.owasp.csrfguard.NewTokenLandingPage	最初にトークンを生成するページ
org.owasp.csrfguard.TokenPerPage	ページごとにトークンを生成する
org.owasp.csrfguard.ProtectedMethods	CSRFから保護するHTTPメソッド
org.owasp.csrfguard.UnprotectedMethods	CSRFから保護しないHTTPメソッド
org.owasp.csrfguard.unprotected.*	CSRFから保護しないファイル
org.owasp.csrfguard.protected.*	CSRFから保護するファイル
org.owasp.csrfguard.action.*	リクエスト受信時の各種動作
org.owasp.csrfguard.TokenName	トークン名
org.owasp.csrfguard.SessionKey	セッションキー名
org.owasp.csrfguard.TokenLength	トークンの長さ
org.owasp.csrfguard.PRNG	疑似乱数生成器の指定(SHA1PRNG等)

OWASP CSRFGuard適用方法(4)

4. JSPファイルのhead要素内またはbody要素の閉じタグ直前にnonceを追加するためのJavaScript(OWASP CSRFGuardのServlet)を指定したscript要素を挿入する

```
<script src="/path/to/webapp/JavaScriptServlet"></script>
```

各対策ライブラリの概要



Spring Security 概要

- 名称： Spring Security
  - プロジェクトURL： http://projects.spring.io/spring-security/
  - ドキュメントURL： http://docs.spring.io/spring-security/site/docs/3.2.5.RELEASE/reference/htmlsingle/#csrf-using
- 概要：
  - 認証認可を提供するServletFilter群のなかにCSRF対策機能も実装されている
  - CSRF対策としてnonceを用いた検証を行う
  - Spring Framework で使うことが前提. 一般的に Spring Security 単体では使えない
- 対策原理：
  - JavaサーブレットのFilter機構を用いてクライアントとサーブレット本体の間に割り込み、リクエストにnonce付加、レスポンス中のnonce検証を行う
- 適用手順：
  1. Webアプリケーションの設定を記載する web.xml に Spring Security を指定する
  2. SpringのXML設定ファイル(applicationSecurity.xml)の http 要素の中にcsrf要素を記載
  3. JSPファイルのform要素にトークンを記載したinput要素を追加する

Spring Security 適用方法(1)

1. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する

```
<listener>
  <listener-class>org.springframework.web.context.contextLoaderListener
  </listener-class>
</listener>
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml /WEB-INF/applicationSecurity.xml
  </param-value>
</context-param>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy
  </filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Spring Security 適用方法(2)

2. SpringのXML設定ファイルのhttp要素の中にcsrf要素を記載
   - Spring 4.xからはデフォルトで有効になっている

```
<http>
   .....
   .....
   <csrf />
</http>
```

Spring Security 適用方法(3)

2. JSPファイルのform要素にトークンを記載したinput要素を追加する

- フォームによる送信の場合

```
<c:url var="logoutUrl" value="/logout"/>
<form action="${logoutUrl}" method="post">
  <input type="submit" value="Log out" />
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
</form>
```

- Ajaxによるリクエストの場合、meta要素にHTTPカスタムヘッダを定義する

```
<head>
    <meta name="_csrf" content="${_csrf.token}"/>
    <!-- default header name is X-CSRF-TOKEN -->
    <meta name="_csrf_header" content="${_csrf.headerName}"/>
（省略）
</head>
```

各対策ライブラリの概要

Apache Tomcat CSRF Prevention Filter

OWASP CSRFGuard

Spring Security


csrf-filter

OWASP CSRFProtector

csrf-magic

Django CSRF Protection middleware

csrf-filter概要

- 名称： csrf-filter
  - プロジェクトURL：https://github.com/dernasherbrezon/csrf-filter
- 概要：
  - CSRF対策を行うServletFilter
  - CSRF対策としてnonceを用いた検証を行っている
  - jarではなくJavaファイル１つが提供されている
- 対策原理：
  - JavaサーブレットのFilter機構を用いてクライアントとサーブレット本体の間に割り込み、リクエストにnonce付加、レスポンス中のnonce検証を行う
- 適用手順：
  1. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する
  2. クライアントへ応答するWebページを生成する JSP ファイルのform要素を変更する

csrf-filter適用方法(1)

1. Webアプリケーションの設定を記載する web.xml に本ライブラリを指定する

```
<filter>
  <!-- csrf-filterの宣言 -->
  <filter-name>csrfFilter</filter-name>
  <filter-class>com.google.code.csrf.StatelessCookieFilter</filter-class>
  <!-- パラメータの設定例。詳細は次ページに記載 -->
  <init-param>
    <param-name>csrfTokenName</param-name>
    <param-value>csrf</param-value>
  </init-param>
  <init-param>
    <param-name>exclude</param-name>
    <param-value>/url1,/url/url2</param-value>
  </init-param>
  <init-param>
    <param-name>excludeGET</param-name>
    <param-value>/url3,/url/url4</param-value>
  </init-param>
  <init-param>
    <param-name>excludeGETStartWith</param-name>
    <param-value>/js/,/css/,/img/</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>csrfFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

csrf-filter適用方法(2)

web.xml で設定できるパラメータ

param-name	説明	デフォルト値
csrfTokenName	トークン名(この項目のみ必須)	—
exclude	CSRF対策を行わないURL	—
excludeGET	CSRF Cookieを生成しないURL。Ajaxのリクエストに用いる。	—
excludeGETStartWith	CSRF Cookieを生成しないURL。servletPath().startsWith()のチェックを行わない (CSS、JavaScript、画像等を指定)	—
cookieMaxAge	Cookieの有効期限(秒)	3600秒

csrf-filter適用方法(3)

2. クライアントへ応答するWebページを生成する JSP ファイルのフォームを変更する

- form要素内に以下のinput要素を追加する

```
<input type="hidden" name="csrf" value="${csrf}">
```

- ただし、ファイルの送信等のmultipart/form-data でデータを送信する場合は上記input要素ではなくform要素のaction属性の値を修正する

```
<form action="/url?csrf=${csrf}" method="POST"
enctype="multipart/form-data">
```

各対策ライブラリの概要

- Apache Tomcat CSRF Prevention Filter
- OWASP CSRFGuard
- Spring Security
- csrf-filter
- **OWASP CSRFProtector**
- csrf-magic
- Django CSRF Protection middleware

OWASP CSRFProtector概要

- 名称： OWASP CSRFProtector
  - https://www.owasp.org/index.php/CSRFProtector_Project
  - https://github.com/mebjas/CSRF-Protector-PHP
- 概要：
  - PHP版とApacheのモジュール版が存在 (今回確認したのはPHP版)
  - CSRF対策としてnonceを用いた検証を行っている
- 対策原理：
  - PHPの関数ob_startを用いてWebアプリケーションからの出力をバッファリングし、nonce を付加する
  - 入力フォームの処理を行うPHPスクリプトファイルの先頭に CSRFProtector コードを追加することで、受け取ったリクエストに含まれるnonceを検証する
    - 参考: http://php.net/manual/ja/function.ob-start.php
- 適用方法：
  1. form要素のあるPHPファイルの先頭部分で本ライブラリを読み込み初期化する。
  2. 設定ファイル /libs/config.php を必要に応じて作成する

OWASP CSRFProtector適用方法(1)

1. form要素のあるPHPファイルの先頭部分で本ライブラリを読み込み、初期化する。

```
include_once __DIR__ .'/libs/csrf/csrfprotector.php';
csrfProtector::init();
```

2. 設定ファイル /libs/config.php を必要に応じて作成する。項目は以下の通り。

OWASP CSRFProtector適用方法(2)

設定キー	説明	デフォルト値
CSRFP_TOKEN	Cookieやフォームに記載するトークンのキー名	"" (空文字。実際には"csrfp_token"と出力される)
noJs	JavaScriptの使用	false (no-js版の場合)
logDirectory	ログ出力ディレクトリ	"../log“ (このディレクトリを書き込み可能にしておく)
failedAuthAction	トークン検証に失敗したときにクライアントへ返すメッセージ • 0: HTTP 403(Forbidden) • 1: GET/POSTクエリを削除して転送($_POSTを削除) • 2: エラーページへ転送(errorRedirectionPage) • 3: エラーメッセージ(customErrorMessageで設定) • 4: HTTP 500(Internal Server Error)	• GET: 0(HTTP 403) • POST: 0(HTTP 403)
errorRedirectionPage	エラーページの絶対URL	"" (空文字)
customErrorMessage	エラーメッセージ	"" (空文字)
jsPath	JavaScriptファイルのconfig.phpからの相対パス	"../js/csrfprotector.js"
jsUrl	JavaScriptファイルの絶対URL	"http://localhost/test/csrf/js/csrfprotector.js"
tokenLength	トークンの長さ	10 (文字)
disabledJavascriptMessage	JavaScriptが無効の際に表示するメッセージ	(省略:このWebサイトはCSRF対策を行っている為JavaScriptを有効にしてくださいという旨の英文)
verifyGetFor	GETリクエストの際に検証を行うURLリスト	「array()」 (空のリスト)

各対策ライブラリの概要

- Apache Tomcat CSRF Prevention Filter
- OWASP CSRFGuard
- Spring Security
- csrf-filter
- OWASP CSRFProtector
- csrf-magic
- Django CSRF Protection middleware

csrf-magic概要・適用方法

- 名称： csrf-magic
  - http://csrf.htmlpurifier.org/
  - https://github.com/ezyang/csrf-magic
- 概要：
  - 既存のPHPスクリプトに本対策ライブラリを読み込むだけで使用可能
  - CSRF対策としてnonceを用いた検証を行っている
- 対策原理：
  - PHPの関数ob_startを用いてWebアプリケーションからの出力をバッファリングし、nonce を付加する
  - 入力フォームの処理を行うPHPスクリプトファイルの先頭に csrf-magic のコードを追加することで、受け取ったリクエストに含まれるnonceを検証する
    - 参考: http://php.net/manual/ja/function.ob-start.php
- 適用方法：

form要素のあるPHPファイルの先頭部分で本ライブラリの読み込みを行うコードを追加する

```
include_once '/path/to/csrf-magic.php';
```

各対策ライブラリの概要

- Apache Tomcat CSRF Prevention Filter
- OWASP CSRFGuard
- Spring Security
- csrf-filter
- OWASP CSRFProtector
- csrf-magic
- Django CSRF Protection middleware

Django CSRF protection middleware概要

- 名称： CSRF Protection middleware
  - https://www.djangoproject.com/
  - https://docs.djangoproject.com/en/1.8/ref/csrf/
- 概要：
  - Pythonで実装されたWebアプリケーションフレームワークDjangoが提供するCSRF対策機能
  - CSRF対策としてnonceとリファラヘッダを用いた検証を行っている
- 対策原理：

  Djangoのmiddlewareはリクエストとレスポンスの処理をフックするための仕組みを提供している。この仕組みを利用して、クライアントに送るフォームにnonceを追加したり、受信した入力フォームに含まれるnonceの検証を行う。
  https://docs.djangoproject.com/en/1.8/topics/http/middleware/
  https://docs.djangoproject.com/en/1.8/ref/middleware/#module-django.middleware.csrf

Django CSRF Protection middleware適用方法(1)

1. ‘django.middleware.csrf.CsrfViewMiddleware’ ミドルウェアを MIDDLEWARE_CLASSES に追加 (設定によっては最初から含まれている)

2. POSTリクエストを送るすべてのform要素内にcsrf_tokenタグを追加する

```
<form action="." method="post">{% csrf_token %}
```

Django CSRF Protection middleware適用方法(2)

3. 上記タグを使用できるようにするため、次のいずれかの方法で対応する画面のビュー内部で‘django.core.context_processors.csrf’コンテキストプロセッサを使用できるようにする
   - RequestContextを使用する
     - RequestContextは 'django.core.context_processors.csrf' を常に使用する
   - ‘django.core.context_processors.csrf’ コンテキストプロセッサをインポートし、CSRF トークンを生成するコードを追加する

```
from django.shortcuts import render_to_response
from django.core.context_processors import csrf
def my_view(request):
    c = {}
    c.update(csrf(request))
    # ... view code here
    return render_to_response("a_template.html", c)
render_to_response(“a_template.html”, c)
```

CSRF対策ライブラリを適用してみよう

CSRF対策ライブラリを適用してみよう

JAVA編: SPACEWALK

Spacewalkとクロスサイトリクエストフォージェリ

製品名：Spacewalk

- 製品概要： Red Hat が提供するLinuxシステムの統合管理ツールRed Hat Network SatelliteのOSS版
- 製品URL：http://spacewalk.redhat.com/

- CVE：CVE-2009-4139
- アドバイザリURL：
  - http://www.redhat.com/support/errata/RHSA-2011-0879.html
  - http://securitytracker.com/id?1025674
  - https://bugzilla.redhat.com/show_bug.cgi?id=529483
  - http://xforce.iss.net/xforce/xfdb/68074
- 脆弱性指摘バージョン：1.2.39
- 脆弱性修正バージョン：1.2.39-85
- 今回適用を行ったSpacewalkは、同じCVEの脆弱性を持つ1.5系で行った。
  - 脆弱性指摘バージョン：1.5.46-1
  - 脆弱性修正バージョン：1.5.47-1

脆弱なコードのポイント

```
<form method="POST" action="/rhn/users/DeleteUserSubmit.do?uid=2">
<div align="right">
      <hr />
      <input type="submit" value="ユーザーの削除" />
    </div>
</form>
```

- 以下の画面では、ユーザーtestuserを削除するフォームが表示されている。

SpacewalkにおけるCSRFの修正

- 脆弱性修正バージョン1.5.47-1では、カスタムタグを定義し、リクエストの内容にトークンを追加している。
  - フォームが置かれたJSPファイル(348個)へカスタムタグを挿入

```
<form method="POST"
      action="/rhn/users/DeleteUserSubmit.do?uid=${param.uid}">
    <rhn:csrf />
    <html:submit styleClass="btn btn-danger">
        <bean:message key="deleteuser.jsp.delete"/>
    </html:submit>
</form>
```

OWASP CSRFGuardの適用

Spacewalk は Java でつくられているため、ここでは Java 向けの CSRF対策ライブラリである OWASP CSRFGuard の適用を試みた。

1. Spacewalkのソースコードを修正
    1. web.xml にフィルタ設定
    2. jsp ファイルに javascript 参照の設定を追加
2. 修正したソースコードでSpacewalkパッケージ作成
3. 作成したパッケージを上書きインストール

JSPファイルへのJavaScript参照の挿入

- code/webapp/WEB-INF/decorators/layout_c.jsp の末尾部分にOWASP CSRFGuardのJavaScriptServletの参照を挿入する。
- layout_c.jsp は各ページのテンプレートとなっており、1か所の変更ですべてのページへ変更を適用できる。

```
      </div><!-- end bottom-wrap -->
    </div><!-- end wrap -->
  <script src="/rhn/JavaScriptServlet"></script>
  </body>
</html:html>
```

ライブラリ適用の効果

- ユーザー削除フォームのform要素のaction属性と、OWASP_CSRFTOKENという名前でinput要素が追加された。

```
<form method="POST"
action="/rhn/users/DeleteUserSubmit.do?uid=3&amp;OWASP_CSRFTOKEN=FW
J8-NCD0-0KUN-3BIV-DGLI-U6UU-1XNT-9K4N">
    <div align="right">
        <hr><input value="ユーザーの削除" type="submit">
    </div>
<input value="FWJ8-NCD0-0KUN-3BIV-DGLI-U6UU-1XNT-9K4N"
name="OWASP_CSRFTOKEN" type="hidden"></form>
```

ライブラリ適用の効果

Spacewalkシステム外のフォームから攻撃リクエストを送信すると、
設定に記載したリダイレクト先であるSpacewalkの403ページ
(http://website/errors/403.html)へ転送される。

※ Tomcatのログ catalina.out にはCSRF攻撃を防いだ情報が記録される。

```
情報: CsrfGuard analyzing request /rhn/users/DeleteUserSubmit.do
2015/02/20 12:53:32 org.owasp.csrfguard.log.JavaLogger log
警告: potential cross-site request forgery (CSRF) attack thwarted
(user:<anonymous>, ip:192.168.80.1, method:POST,
uri:/rhn/users/DeleteUserSubmit.do, error:required token is missing from
the request)
```

SpacewalkへのOWASP CSRFGuard適用の考察(1)

- Spacewalk自身が行ったCSRF対策は、nonceを用いるクラスの追加とフォームを含むJSPファイルの修正 (JSPファイル348個の修正を行っている)

- OWASP CSRFGuardを用いたCSRF対策では、nonceの挿入をJavaScriptで行うため、各ページが共通で使用するJSPファイル1つのみを修正することで適用できる
  - ただし、OWASP CSRFGuardの適用後はJavaScriptを使用できないブラウザではリクエストが拒否されるため、使用環境の条件に注意。

- Spacewalkには独自の403ページがあったため、OWASP CSRFGuardの設定にてリクエスト拒否時は403ページへ転送するようにし、Spacewalkシステムに合った動作を行うように設定できた

SpacewalkへのOWASP CSRFGuard適用の考察(2)

- SpacewalkがStrutsを使用している事もあり、OWASP CSRFGuardのドキュメント通りの適用方法では動作しなかった
  - StrutsはActionに用いるJSPの設定があり、JSPファイル名がURLに含まれないため、修正対象のJSPを確認する必要があった
  - Tomcatのディレクトリへ配置されたJSPファイルを修正しても、実際に実行されるのは事前にコンパイルされたJSPファイルのため、修正内容は反映されない
  - そのため、ソースコードを修正したSpacewalkのパッケージの作成を行い、パッケージをインストールしてTomcatのディレクトリへ配置する必要があった

- 以上から、 SpacewalkへのOWASP CSRFGuardの適用は、Strutsの構造を理解している前提で、Spacewalkが行ったCSRF対策と同等またはそれ以下のコストでCSRF対策が行えることが分かった。

CSRF対策ライブラリを適用してみよう

PHP編: LOGIN REBUILDER

PHPで実装された製品へのCSRF対策ライブラリの適用

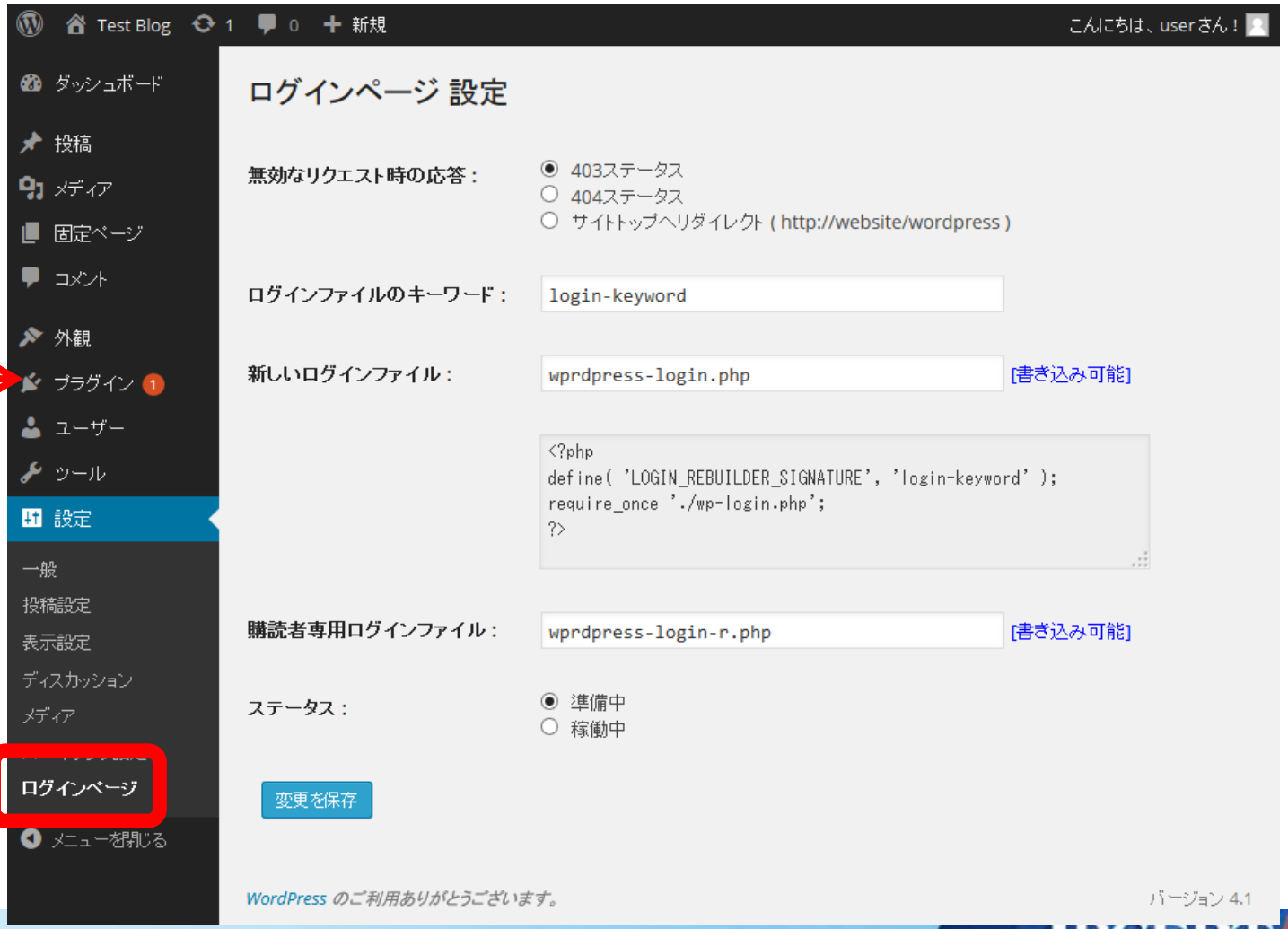
製品名：Login Rebuilder

- 製品概要：ログインページをデフォルトのwp-login.phpから変更する事でセキュリティを向上するWordPressプラグイン。
- 製品URL：
  - https://wordpress.org/plugins/login-rebuilder/
  - http://plugins.svn.wordpress.org/login-rebuilder
- CVE：CVE-2014-3882
- アドバイザリURL：
  - http://wordpress.org/plugins/login-rebuilder/changelog/
  - http://12net.jp/news/n20140623_01.html
  - https://jvn.jp/jp/JVN05329568/index.html
- 脆弱性指摘バージョン：1.1.3(リビジョン868421)
- 脆弱性修正バージョン：1.2.0(リビジョン914619)

脆弱なフォームの場所

- WordPress管理画面のLogin Rebuilderの設定フォーム(「設定」→「ログインページ」)
- http://website/wordpress/wp-admin/options-general.php?page=login-rebuilder-properties

脆弱なバージョンは最新ではないため、更新があることを知らせる赤い数字が表示されている。



formタグ内の内容

```
<form method="post" action="/wordpress/wp-admin/options-general.php?page=login-rebuilder-
properties">
(省略)
<input type="radio" name="properties[response]" id="properties_response_1"
value="1" checked='checked' />
<input type="radio" name="properties[response]" id="properties_response_2" value="2" />
<input type="radio" name="properties[response]" id="properties_response_3" value="3" />
(省略)
<input type="text" name="properties[keyword]" id="properties_keyword" value="login-keyword"
class="regular-text code" />
(省略)
<input type="text" name="properties[page]" id="properties_page" value="wprdpress-login.php"
class="regular-text code" />
(省略)
<textarea  name="properties[content]" id="login_page_content" rows="4" cols="60" style="font-
family:monospace;" readonly="readonly"></textarea>
(省略)
<input type="text" name="properties[page_subscriber]" id="properties_page_subscriber"
value="wprdpress-login-r.php" class="regular-text code" />
(省略)
<input type="radio" name="properties[status]" id="properties_status_0" value="0"
checked='checked' />
<input type="radio" name="properties[status]" id="properties_status_1" value="1"  />
(省略)
<input type="submit" name="submit" value="変更を保存" class="button-primary" />
</form>
```

Login RebuilderのCSRF修正バージョンのポイント

- WordPressのwp_create_nonce()を使用して一時的な値 $nonce を生成し、フォームの既存のキー(properties[response])に追加。

- nonce生成部分

```
$nonce = wp_create_nonce(
self::LOGIN_REBUILDER_PROPERTIES_NAME.'@'.$wp_version.'@'.LOGIN_REBUILDER_DB_VERSION);
```

- input要素のname属性の値への追加例 (properties[page]の場合、他も同様)

```
<input type="text" name="properties_<?php echo $nonce; ?>[page]" id="properties_page"
value="<?php _e( $this->properties['page'] ); ?>" class="regular-text code" />
```

- nonceの検証部分(POST内容のキー名の存在確認)

```
if ( isset( $_POST['properties_'.$nonce] ) ) {
```

Login RebuilderのCSRF修正バージョンのポイント

- 修正後のHTMLのinput要素 (properties[page]の場合)

```
<input type="text" name="properties_c6808428a6[page]" id="properties_page"
value="wprdpress-login.php" class="regular-text code" />
```

- inputのname属性を「properties[response]」から「properties_**c6808428a6**[response]」といった一時的なものを使用するように変更
- フォームからPOSTされた内容のキーがproperties_**c6808428a6**ではない場合は変更を行わない

Login Rebuilderのnonceを用いたCSRF対策の実装は、独立したinput要素ではなく既存のinput要素のname属性の値を変更するという形で行われている。
(今回確認したバージョン1.2.0の場合)

OWASP CSRF Protector の適用

Login Rebuilder は PHP でつくられているため、ここでは PHP 向けの CSRF対策ライブラリである OWASP CSRF Protector の適用を試みた。

1. OWASP CSRF Protector のソースコードを Login Rebuilder から参照できる場所に置く
2. Login Rebuilder のソースコードを修正、CSRF Protector を読み込むようにする

OWASP CSRF Protectorの設置

WordPress プラグインディレクトリ内の Login Rebuilder インストールディレクトリ内に OWASP CSRF Protector を設置。

```
wordpress/                      ←WordPressインストールディレクトリ
+ wp-content/plugins/               ←WordPressプラグインディレクトリ
  + login-rebuilder/                ←Login Rebuilderインストールディレクトリ
    + languages/                    ←言語ファイルディレクトリ(中身省略)
    + csrfp/                        ←OWASP CSRF Protector設置ディレクトリ
      + js/
        + csrfprotector.js          ←トークン挿入用のJavaScript
        + index.php
      + libs/
        + csrf/                     ←OWASP CSRF Protector本体ディレクトリ
          + csrfpJsFileBase.php
          + csrfprotector.php
          + index.php
        + config.sample.php         ←設定ファイルサンプル
        + index.php
      + log/                        ←ログ出力ディレクトリ
        + .htaccess
    + login-rebuilder.php           ←Login Rebuilder本体
    + uninstall.php
```

Login Rebuilder本体の編集

- Login Rebuilder の本体 login-rebuilder.php を編集し、OWASP CSRF Protector の csrfprotector.php を読み込む
- 先頭のdefineブロックの下に記述

```
（省略）
define( 'LOGIN_REBUILDER_DOMAIN', 'login-rebuilder' );
define( 'LOGIN_REBUILDER_DB_VERSION_NAME', 'login-rebuilder-db-version' );
define( 'LOGIN_REBUILDER_DB_VERSION', '1.1.3' );
define( 'LOGIN_REBUILDER_PROPERTIES', 'login-rebuilder' );

// 追加:Login Rebuilderインストールパスの定義
define( 'LOGIN_REBUILDER_PLUGIN_DIR', plugin_dir_path( __FILE__ ) );
// 追加:OWASP CSRF Protectorの読み込みと初期化
require_once( LOGIN_REBUILDER_PLUGIN_DIR . 'csrfp/libs/csrf/csrfprotector.php' );
csrfProtector::init();

$plugin_login_rebuilder = new login_rebuilder();
（省略）
```

CSRF Protector 適用の効果

- 出力されるHTMLでは、form要素内にnonceが設定されたcsrfp_tokenという名前のinput要素が追加される。

```
<form method="post" action="/wordpress/wp-admin/options-
general.php?page=login-rebuilder-properties">
（省略）
<input type='hidden' name='csrfp_token' value='559e21982a' />
```

- 攻撃リクエストを送信すると、エラーページへ転送を行う設定の通り、WordPressブログのトップページ (http://website/wordpress/)へ転送される。
- ログディレクトリに追加されたファイルにはCSRF攻撃のリクエストの内容が記録される。

```
{"timestamp":1423729288,"HOST":"192.168.80.129","REQUEST_URI":"¥/wordpre
ss¥/wp-admin¥/options-general.php?page=login-rebuilder-
properties","requestType":"POST",
"query":{"properties":{"response":"3","keyword":"CSRF","page":"csrf.php"
,"content":"","page_subscriber":"csrf2.php","status":"1"},
"submit":"¥u5909¥u66f4¥u3092¥u4fdd¥u5b58"},"cookie":[]}
```

Login RebuilderへのOWASP CSRF Protector適用の考察(1)

- Login Rebuilder自身がバージョン1.2.0で行ったCSRF対策は、nonce情報を通常のフォーム内容のキーの一部として使用するというもの
  - WordPress本体はCSRF対策に利用できるnonceを取り扱う関数を提供しているが、 Login Rebuilderはnonce生成関数のみを使用しており、検証関数は使用せずに独自の実装を行っていた

- OWASP CSRF Protectorを用いたCSRF対策は、フォームを含むPHPファイルへOWASP CSRF Protectorを読み込ませる数行の修正および11項目の設定の編集のみで行えた

Login RebuilderへのOWASP CSRF Protector適用の考察(2)

- OWASP CSRF Protector はWordPressのプラグインのようなサブシステムへの適用も可能であることが分かった
  - OWASP CSRF Protectorの設定を適切に行う事により、リクエスト拒否時にはWordPressのブログトップページへ転送するようにした
  - デフォルトの設定では、リクエスト拒否時にHTTP 403レスポンスとしてWordPressの画面ではない白地に黒文字でリクエスト拒否の旨が表示される
- Login Rebuilderの修正量および設定の少なさから、新たにnonceを取り扱うように変更を行ったLogin Rebuilderが行ったCSRF対策よりも低いコストでCSRF対策が行えた、と言っていいのではないか?

まとめ

まとめ(1)

- CSRF(クロスサイトリクエストフォージェリ)という脆弱性の存在とその適切な対策を理解する

- CSRF対策の実装方法を理解し実践する
  - 独自に実装する
  - WebフレームワークのCSRF対策機能を活用する
  - CSRF対策ライブラリを活用する

まとめ(2)

- CSRF対策ライブラリについて
  - JavaやPHPにおけるCSRF対策ライブラリの実装アプローチの理解
  - CSRF対策ライブラリの適用を試行
    - SpacewalkへOWASP CSRFGuard
    - Login RebuilderへOWASP CSRF Protector

CSRF対策ライブラリを的確に適用するには、Webアプリケーションフレームワークの構成とCSRF対策の原理を理解している必要がある

本資料では既存のWebアプリケーションへのCSRF対策ライブラリ適用の試みを紹介した。
新規にWebアプリケーションを開発する際においても、CSRF対策ライブラリの活用を検討する価値はある。

参考情報

- 独立行政法人 情報処理推進機構(IPA)
  - 安全なウェブサイトの作り方
    (https://www.ipa.go.jp/security/vuln/websecurity.html)
  - セキュアプログラミング講座、Webアプリケーション編
    (https://www.ipa.go.jp/security/awareness/vendor/programmingv2/web.html)
- OWASP
  - Cross-Site Request Forgery (CSRF)
    (https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29)
  - Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
    (https://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet)
- Wikipedia
  - (英語版) Cross-site request forgery
    https://en.wikipedia.org/wiki/Cross-site_request_forgery
  - (日本語版) クロスサイトリクエストフォージェリ
    https://ja.wikipedia.org/wiki/クロスサイトリクエストフォージェリ

著作権・引用や二次利用について

■本資料の著作権はJPCERT/CCに帰属します。

■本資料あるいはその一部を引用・転載・再配布する際は、引用元名、資料名および URL の明示をお願いします。

記載例
引用元：一般社団法人JPCERTコーディネーションセンター
クロスサイトリクエストフォージェリ(CSRF)とその対策
https://www.jpcert.or.jp/securecoding/AntiCSRF-201510.pdf

■本資料を引用・転載・再配布をする際は、引用先文書、時期、内容等の情報を、JPCERT コーディネーションセンター広報(pr@jpcert.or.jp)までメールにてお知らせください。なお、この連絡により取得した個人情報は、別途定めるJPCERT コーディネーションセンターの「プライバシーポリシー」に則って取り扱います。

本資料の利用方法等に関するお問い合わせ
JPCERTコーディネーションセンター
広報担当
E-mail：pr@jpcert.or.jp

本資料の技術的な内容に関するお問い合わせ
JPCERTコーディネーションセンター
セキュアコーディング担当
E-mail：secure-coding@jpcert.or.jp