Webシステム／Webアプリケーションセキュリティ要件書 3.0

January 2019

1. Webシステム／Webアプリケーションセキュリティ要件書について

1.1. 本ドキュメントについて

Webシステム／Webアプリケーションセキュリティ要件書（以下、本ドキュメント）は、安全なWebアプリケーションの開発に必要なセキュリティ要件書です。発注者、開発者、テスト実施者、セキュリティ専門家、消費者が活用することで、以下のことを達成することを目的としています。

- 開発会社・開発者に安全なWebシステム／Webアプリケーションを開発してもらうこと
- 開発会社と発注者の瑕疵担保契約の責任分解点を明確にすること
- 要求仕様やRFP（提案依頼書）として利用し、要件定義書に組み込むことができるセキュリティ要件として活用していただくこと

1.2. 本ドキュメントがカバーする範囲

本ドキュメントではWebシステム／Webアプリケーションに関して一般的に盛り込むべきだと考えられるセキュリティ要件について記載しています。また、開発言語やフレームワークなどに依存することなくご利用いただけます。ただし、ネットワークやホストレベル、運用などに関するセキュリティ要件については記載していません。

対象とするWebシステム／Webアプリケーションは、インターネット・イントラネット問わず公開するシステムで、特定多数または不特定多数のユーザーが利用するシステムを想定しています。この中でも特に認証を必要とするシステムが、本ドキュメントの主なターゲットとなっています。

本ドキュメントは、セキュリティ要件としての利用しやすさを優先して記載しているため、一般的であろうというシステムを想定し、例外の記載を少なくしたセキュリティ要件となっています。そのため具体的な数値や対策を指定していることもありますが、要件定義書に記載する内容は開発者と折衝してください。

2. about OWASP

2.1. OWASPについて

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。OWASPでは、以下をフリーでオープンな形で提供・実施しています。（https://www.owasp.org/）

- アプリケーションセキュリティに関するツールと規格
- アプリケーションセキュリティ検査、セキュア開発、セキュリティ・コードレビューに関する網羅的な書籍
- 標準のセキュリティ制御とライブラリ
- 世界中の支部
- 先進的な研究
- 世界中での会議
- メーリング・リスト

全てのOWASPのツール、文書、フォーラム、および各支部は、アプリケーションセキュリティの改善に関心を持つ人のため、無料で公開されています。アプリケーションセキュリティに対する最も効果的なアプローチとして、我々は人、プロセス、技術という3つの課題から改善することを提唱しています。

OWASPは新しい種類の組織です。商業的な圧力が無い中、アプリケーションセキュリティに対して、偏見無く実用的かつコスト効果の高い情報の提供を行っています。OWASP　はいかなるIT企業の支配下にもありませんが、商用のセキュリティ技術の活用を支持しています。他のオープンソース・ソフトウェアプロジェクトと同様に、OWASPも協同かつオープンな形で多様な資料を作成しています。

The OWASP Foundation は、このプロジェクトの長期的な成功を目指す非営利組織です。OWASP理事会、グローバル委員会、支部長、プロジェクトリーダー、プロジェクトメンバーを含む、OWASPの関係者はほとんどボランティアです。革新的なセキュリティ研究に対して、助成金とインフラの提供で支援しています。

2.2. OWASP Japanについて

主に日本で活動しているOWASPメンバーによる日本支部です。OWASPの膨大なドキュメントやツール類の日本語化を初めとして、日本からのセキュリティ情報の発信を行っています。是非ご参加下さい。
https://www.owasp.org/index.php/Japan

3. 要求仕様項目

「※」：必須事項ではないが、あると望ましい要件を表しています。囲み内は解説および補足です。

1. 認証・認可

1.1 ユーザー認証について

- 特定のユーザーのみに表示・実行を許可すべき画面や機能、APIでは、ユーザー認証を実施すること
- 上記画面や機能に含まれる画像やファイルなどの個別のコンテンツでは、ユーザー認証を実施すること
  （非公開にすべきデータは直接URLで指定できる公開ディレクトリに配置しない）
- 管理者用画面では、ユーザー認証を実施すること

特定のユーザーのみにアクセスを許可したいWebシステムでは、ユーザー認証を行う必要があります。
また、ユーザー認証が成功した後にはアクセス権限を確認する必要があります。そのため、認証済みユーザーのみがアクセス可能な箇所を明示しておくことが望ましいでしょう。
リスクベース認証や二要素認証など認証をより強固にする仕組みもあります。

1.2 ユーザーの再認証について ※

- 個人情報や機微情報を表示するページに遷移する際には、再認証を実施すること
- パスワード変更や決済処理などの重要な機能を実行する際には、再認証を実施すること

ユーザー認証はセッションにおいて最初の一度だけ実施するのではなく、重要な情報や機能へアクセスする際には再認証を行うことが望ましいでしょう。

1.3 パスワードについて

- パスワード文字列は少なくとも大小英字と数字の両方を含み、最低 8文字以上であること
- 画面（hiddenフィールドなどのHTMLソース内も含む）にパスワード文字列を表示しないこと
- パスワード文字列の入力フォームはinput type="password"で指定すること
- ユーザーが入力したパスワード文字列を次画面以降で表示しないこと
- パスワードは「パスワード文字列＋salt（ユーザー毎に異なるランダムな文字列）」をハッシュ化したものとsaltのみを保存すること（saltは20文字以上であることが望ましい）
- ユーザー自身がパスワードを変更できる機能を用意すること ※
- 登録可能なパスワード文字列の最大文字数は127文字以上であること ※
- パスワード文字列として使用可能な文字種は制限しないこと ※
  (任意の大小英字、数字、記号、空白などが利用可能であること)

認証を必要とするWebシステムの多くは、パスワードを本人確認の手段として認証処理を行います。そのためパスワードを盗聴や盗難などから守ることが重要になります。
パスワード文字列のハッシュ化をさらに安全にする手法としてストレッチングがあります。

1.4 アカウントロック機能について

- 認証時に無効なパスワードで10回試行があった場合、最低30分間はユーザーがロックアウトされた状態にすること
- ロックアウトは自動解除を基本とし、手動での解除は管理者のみ実施可能とすること ※

パスワードに対する総当たり攻撃や辞書攻撃などから守るためには、試行速度を遅らせるアカウントロック機能の実装が有効な手段になります。アカウントロックの試行回数、ロックアウト時間については、サービスの内容に応じて調整することが必要になります。

1.5 パスワードリセット機能について

- パスワードリセットを実行する際にはユーザー本人しか受け取れない連絡先（あらかじめ登録しているメールアドレス、電話番号など）に再設定方法を通知すること
- パスワードはユーザー自身に再設定させること

連絡先については、事前に受け取り確認をしておくことでより安全性を高めることができます

1.6 アクセス制御について

- Web ページや機能、データにアクセスする際には認証情報・状態を元に権限があるかどうかを判別すること
- Web ページや機能、データをアクセス制御（認可制御）する際には認証情報・状態を元に権限があるかどうかを判別すること

認証により何らかの制限を行う場合には、利用しようとしている情報や機能へのアクセス（読み込み・書き込み・実行など）権限を確認することでアクセス制御を行うことが必要になります。

画像やファイルなどのコンテンツ、APIなどの機能に対しても、全て個別にアクセス権限を設定、確認する必要があります。
これらはアクセス権限の一覧表に基づいて行います。
CDNなどを利用してコンテンツを配置するなどアクセス制御を行うことが困難な場合、予測が困難なURLを利用することでアクセスされにくくする方法もあります。

2. セッション管理

2.1 セッションの破棄について

- 認証済みのセッションが一定時間以上アイドル状態にあるときはセッションタイムアウトとし、サーバー側のセッションを破棄しログアウトすること
- ログアウト機能を用意し、ログアウト実行時にはサーバー側のセッションを破棄すること

認証を必要とするWebシステムの多くは、認証状態の管理にセッションIDを使ったセッション管理を行います。認証済みの状態にあるセッションを不正に利用されないためには、使われなくなったセッションを破棄する必要があります。セッションタイムアウトの時間については、サービスの内容やユーザー利便性に応じて設定することが必要になります。
ログアウト機能の実行後にその成否をユーザーが確認できることが望ましい。

2.2 セッションIDについて

- Webアプリケーションフレームワークなどが提供するセッション管理機能を使用すること
- セッションIDの発行は認証成功後とする、または、認証前にセッションIDを発行している場合は、認証成功直後に新たなセッションIDを発行すること
- ログイン前に機微情報をセッションに格納する時点でセッションIDを発行または再生成すること
- 認証済みユーザーの特定はセッションに格納した情報を元に行うこと

セッションIDを用いて認証状態を管理する場合、セッションIDの盗聴や推測、攻撃者が指定したセッションIDを使わされるなどの攻撃から守る必要があります。
また、セッションIDは原則としてcookieにのみ格納すべきです。

2.3 CSRF（クロスサイトリクエストフォージェリー）対策の実施について

- ユーザーにとって重要な処理を行う箇所では、ユーザー本人の意図したリクエストであることを確認できるようにすること

正規ユーザー以外の意図により操作されては困る処理を行う箇所では、フォーム生成の際に他者が推測困難なランダムな値（トークン）をhiddenフィールドに埋め込み、リクエストをPOSTメソッドで送信します。フォームデータを処理する際にトークンが正しいことを確認することで、正規ユーザーの意図したリクエストであることを確認することができます。
また、別の方法としてパスワード再入力による再認証を求める方法もあります。

3. パラメーター

3.1 URLにユーザーID やパスワードなどの機微情報を格納しないこと

URLは、リファラー情報などにより外部に漏えいする可能性があります。そのため URL には秘密にすべき情報は格納しないようにする必要があります。

3.2 パラメータ（クエリーストリング、エンティティボディ、cookieなどクライアントから受け渡される値）にパス名を含めないこと

ファイル操作を行う機能などにおいて、URL パラメータやフォームで指定した値でパス名を指定できるようにした場合、想定していないファイルにアクセスされてしまうなどの不正な操作を実行されてし まう可能性があります。

3.3 パラメーター要件に基づいて、入力値の文字種や文字列長の検証を行うこと

各パラメーターは、機能要件に基づいて文字種・文字列長・形式を定義する必要があります。入力値に想定している文字種や文字列長以外の値の入力を許してしまう場合、不正な操作を実行されてしまう可能性があります。サーバー側でパラメータを受け取る場合、クライアント側での入力値検証の有無に関わらず、入力値の検証はサーバー側で実施する必要があります。

3.4 入力値としてファイルを受け付ける場合には、拡張子やファイルフォーマットなどの検証を行うこと

ファイルのアップロード機能を利用した不正な実行を防ぐ必要があります。

4. 出力処理

4.1 HTMLを生成する際の処理について

- HTMLとして特殊な意味を持つ文字（< > " ' &）を文字参照によりエスケープすること
- 外部から入力したURLを出力するときは「http://」または「https://」で始まるもののみを許可すること
- <script>...</script>要素の内容やイベントハンドラ（onmouseover=""など）を動的に生成しないようにすること
- 任意のスタイルシートを外部サイトから取り込めないようにすること

外部からの入力により不正なHTMLタグなどが挿入されてしまう可能性があります。「<」→「<」や「&」→「&」、「"」→「"」のようにエスケープを行う必要があります。スクリプトによりクライアント側でHTMLを生成する場合も、同等の処理が必要です。実装の際にはこれらを自動的に実行するフレームワークやライブラリを使用することが望ましいでしょう。また、その他にもスクリプトの埋め込みの原因となるものを作らないようにする必要があります。
<script>...</script>要素の内容やイベントハンドラは原則として動的に生成しないようにすべきですが、jQueryなどのAjaxライブラリを使用する際はその限りではありません。ライブラリについては、アップデート状況などを調べて信頼できるものを選択するようにしましょう。
XMLを生成する場合も同様にエスケープが必要です。

4.2 HTMLタグの属性値を「"」で囲うこと

HTMLタグ中のname="value"で記される値(value)にユーザーの入力値を使う場合、「"」で囲わない場合、不正な属性値を追加されてしまう可能性があります。

4.3 CSSを動的に生成しないこと

外部からの入力により不正なCSSが挿入されると、ブラウザに表示される画面が変更されたり、スクリプトが埋め込まれる可能性があります。

4.4 JSONを生成する際の処理について、文字列連結でJSON文字列を生成せず、適切なライブラリを用いてオブジェクトをJSONに変換すること

適切なライブラリがない場合は、JSONとして特殊な意味を持つ文字（ " \ , : { } [ ] ）をUnicodeエスケープする必要があります。

4.5 HTTPレスポンスヘッダーのContent-Typeを適切に指定すること

一部のブラウザではコンテンツの文字コードやメディアタイプを誤認識させることで不正な操作が行える可能性があります。これを防ぐためには、HTTPレスポンスヘッダーを「Content-Type: text/html; charset=utf-8」のように、コンテンツの内容に応じたメディアタイプと文字コードを指定する必要があります。

4.6 HTTPレスポンスヘッダーフィールドの生成時に改行コードが入らないようにすること

HTTPヘッダーフィールドの生成時にユーザーが指定した値を挿入できる場合、改行コードを入力することで不正なHTTPヘッダーやコンテンツを挿入されてしまう可能性があります。これを防ぐためには、HTTPヘッダーフィールドを生成する専用のライブラリなどを使うようにすることが望ましいでしょう。

4.7 SQL文を組み立てる際に静的プレースホルダを使用すること

SQL文の組み立て時に不正なSQL文を挿入されることで、SQLインジェクションを実行されてしまう可能性があります。これを防ぐためにはSQL文を動的に生成せず、プレースホルダを使用してSQL文を組み立てるようにする必要があります。
静的プレースホルダとは、JIS/ISOの規格で「準備された文(Prepared Statement)」と規定されているものです。

4.8 プログラム上でOSコマンドやアプリケーションなどのコマンド、シェル、eval()などによるコマンドの実行を呼び出して使用しないこと

コマンド実行時にユーザーが指定した値を挿入できる場合、外部から任意のコマンドを実行されてしま

う可能性があります。コマンドを呼び出して使用しないことが望ましいでしょう。

4.9 リダイレクタを使用する場合には特定のURLのみに遷移できるようにすること

リダイレクタのパラメーターに任意のURLを指定できる場合（オープンリダイレクタ）、攻撃者が指定した悪意のあるURLなどに遷移させられる可能性があります。

4.10 メールヘッダーフィールドの生成時に改行コードが入らないようにすること

メールの送信処理にユーザーが指定した値を挿入できる場合、不正なコマンドなどを挿入されてしまう可能性があります。これを防ぐためには、不正な改行コードを使用できないメール送信専用のライブラリなどを使うようにすることが望ましいでしょう。

5. HTTPS

5.1 Webサイトを全てHTTPSで保護すること

適切にHTTPSを使うことで通信の盗聴・改ざん・なりすましから情報を守ることができます。次のような重要な情報を扱う画面や機能ではHTTPSで通信を行う必要があります。

・入力フォームのある画面
・入力フォームデータの送信先
・重要情報が記載されている画面
・セッションIDを送受信する画面

HTTPSの画面内で読み込む画像やスクリプトなどのコンテンツについてもHTTPSで保護する必要があります。

5.2 サーバー証明書はアクセス時に警告が出ないものを使用すること

HTTPSで提供されているWebサイトにアクセスした場合、Webブラウザから何らかの警告がでるということは、適切にHTTPSが運用されておらず盗聴・改ざん・なりすましから守られていません。適切なサーバー証明書を使用する必要があります。

5.3 TLS1.2以上のみを使用すること

SSL2.0／3.0、TLS1.0／1.1には脆弱性があるため、無効化する必要があります。使用する暗号スイートは、8.2を参照してください。

5.4 レスポンスヘッダーにStrict-Transport-Securityを指定すること

Hypertext Strict Transport Security(HSTS)を指定すると、ブラウザがHTTPSでアクセスするよう強制できます。

6. cookie

6.1 cookieの属性を適切に設定すること

- Secure属性を付けること
- HttpOnly属性を付けること
- Domain属性を指定しないこと　※

Secure属性を付けることで、http://でのアクセスの際にはcookieを送出しないようにできます。特に認証状態に紐付けられたセッションIDを格納する場合には、Secure属性を付けることが必要です。
HttpOnly属性を付けることで、クライアント側のスクリプトからcookieへのアクセスを制限することができます。
セッションフィクセイションなどの攻撃に悪用されることがあるため、Domain属性は特に必要がない限り指定しないことが望ましいでしょう。

7. 画面設計

7.1 ユーザーのWebブラウザの設定を変更させずデフォルト状態で動作すること

- ユーザーに対して、セキュリティ設定の変更をさせるような指示をしないこと
- ユーザーに対して、セキュリティ警告を無視させるような指示をしないこと

ユーザーのWebブラウザのセキュリティ設定などを変更した場合や、認証局の証明書をインストールさせる操作は、他のサイトにも影響します。
ブラウザの出す警告を通常利用においても無視させるよう指示をしていると、悪意のあるサイトで同様の指示をされた場合もそのような操作をしてしまう可能性が高まります。

7.2 レスポンスヘッダーにX-Frame-Optionsを指定すること

クリックジャッキング攻撃に悪用されることがあるため、X-Frame-Optionsヘッダーフィールドに DENY またはSAMEORIGINを指定する必要があります。

8. その他

8.1 エラーメッセージに詳細な内容を表示しないこと

ミドルウェアやデータベースのシステムが出力するエラーには、攻撃のヒントになる情報が含まれているため、エラーメッセージの詳細な内容はエラーログなどに出力するべきです。

8.2 ハッシュ関数、暗号アルゴリズムは『電子政府における調達のために参照すべき暗号のリスト（CRYPTREC暗号リスト）』に記載のものを使用すること

広く使われているハッシュ関数、疑似乱数生成系、暗号アルゴリズムの中には安全でないものもあります。安全なものを使用するためには、『電子政府における調達のために参照すべき暗号のリスト（CRYPTREC暗号リスト）』に記載されたものを使用する必要があります。

8.3 鍵や秘密情報などに使用する乱数的性質を持つ値を必要とする場合には、暗号学的な強度を持った疑似乱数生成系を使用すること

鍵や秘密情報に予測可能な乱数を用いると、過去に生成した乱数値から生成する乱数値が予測される可能性があるため、ハッシュ関数などを用いて生成された暗号学的な強度を持った疑似乱数生成系を使用する必要があります。

8.4 公開ディレクトリには公開を前提としたファイルのみ配置すること

公開ディレクトリに配置したファイルは、URLを直接指定することでアクセスされる可能性があります。そのため、機微情報や設定ファイルなどの公開する必要がないファイルは、公開ディレクトリ以外に配置する必要があります。

8.5 基盤ソフトウェアはアプリケーションの稼働年限以上のものを選定すること

脆弱性が発見された場合、修正プログラムを適用しないと悪用される可能性があります。そのため、言語やミドルウェア、ソフトウェアの部品などの基盤ソフトウェアは稼働期間またはサポート期間がアプリケーションの稼働期間以上のものを利用する必要があります。もしアプリケーションの稼働期間中に基盤ソフトウェアの保守期間が終了した場合、危険な脆弱性が残されたままになる可能性があります。

8.6 既知の脆弱性のないOSやミドルウェア、ライブラリやフレームワーク、パッケージなどのコンポーネントを使用すること

8.7 管理者がアカウントの有効・無効を設定できること ※

不正にアカウントを利用されていた場合に、アカウントを無効化することで被害を軽減することができます。

8.8 重要な処理が行われたらログを記録すること

ログは、情報漏えいや不正アクセスなどが発生した際の検知や調査に役立つ可能性があります。認証の失敗やアカウント情報の変更などの重要な処理が実行された場合には、その処理の内容やクライアントのIPアドレスなどをログとして記録することが望ましいでしょう。ログに機微情報が含まれる場合にはログ自体の取り扱いにも注意が必要になります。

8.9 重要な処理が行われたらユーザーに通知すること　※

重要な処理（パスワードの変更など、ユーザーにとって重要で取り消しが困難な処理）が行われたことをユーザーに通知することによって異常を早期に発見できる可能性があります。

8.10 XMLを読み込む際は、外部参照を無効にすること

8.11 Access-Control-Allow-Originヘッダーを指定する場合は、動的に生成せず固定値を使用すること

クロスオリジンでXHRを使う場合のみこのヘッダーが必要です。不要な場合は指定する必要はありませんし、指定する場合も特定のオリジンのみを指定する事が望ましいです。

8.12 信頼できないデータ供給元からのシリアライズされたオブジェクトを受け入れないこと

デシリアライズする場合は、シリアライズしたオブジェクトにデジタル署名などを付与し、信頼できる供給元が発行したデータであるかを検証して下さい。

9. 提出物

9.1 提出物として次の資料を用意すること

- サイトマップ
- 画面遷移図
- アクセス権限一覧表
- コンポーネント一覧　※

認証や再認証、CSRF対策が必要な箇所、アクセス制御が必要なデータを明確にするためには、Webサイト全体の構成を把握し、扱うデータを把握する必要があります。そのためには上記の資料を用意することが望ましいでしょう。
誰にどの機能の利用を許可するかまとめた一覧表を作成することが望ましいでしょう。
依存しているライブラリやフレームワーク、パッケージなどのコンポーネントに脆弱性が存在する場合がありますので、依存しているコンポーネントを把握しておく必要があります。

4. 参考文献

- 独立行政法人 情報処理推進機構
  『安全なウェブサイトの作り方　改訂第7版』
  ➢ http://www.ipa.go.jp/security/vuln/websecurity.html
- 株式会社トライコーダ
  『セキュアWebアプリケーション開発講座』
  ➢ https://www.tricorder.jp/training/secure_webapp_dev/
- 書籍『体系的に学ぶ　安全なWebアプリケーションの作り方　脆弱性が生まれる原理と対策の実践（ソフトバンククリエイティブ）』徳丸浩 著
- 総務省「電子政府における調達のために参照すべき暗号のリスト（CRYPTREC暗号リスト）
  ➢ http://www.soumu.go.jp/menu_news/s-news/01ryutsu03_02000038.html

制作：脆弱性診断士スキルマッププロジェクト《特定非営利活動法人日本ネットワークセキュリティ協会の日本セキュリティオペレーション事業者協議会（ISOG-J）のセキュリティオペレーションガイドラインWG（WG1）とOWASP Japan主催の共同ワーキンググループ》

執筆メンバー：
上野 宣　(ISOG-J WG1リーダー／OWASP Japan Chapter Leader／ 株式会社トライコーダ)
国分 裕　(ISOG-J WG1サブリーダー／三井物産セキュアディレクション株式会社)
池田　雅一　(テクマトリックス株式会社)
岩井 基晴　(株式会社イエラエセキュリティ)
大塚 純平　(サイボウズ株式会社)
大塚 淳平　(NRIセキュアテクノロジーズ株式会社)
小河 哲之　(三井物産セキュアディレクション株式会社)
亀田 勇歩　(SCSK株式会社)
洲崎 俊　(三井物産セキュアディレクション株式会社)
関根 鉄平　(株式会社ユービーセキュア)
田中 悠一郎　(NRIセキュアテクノロジーズ株式会社)
長友 比登美　(サイボウズ株式会社)
野口 睦夫　(日本電気株式会社)
八木橋 優
山崎 圭吾　(株式会社ラック)
吉田 聡　(株式会社ラック)

5. 改訂履歴

株式会社トライコーダ版
Ver.1.0　初版（2009年3月17日）
Ver.1.1　2009年4月22日改訂

OWASP版
Ver.1.0　初版（2013年11月1日）
Ver.2.0　2015年10月13日改訂
Ver.3.0　2019年1月15日改訂