

安全な ウェブサイトの 作り方

改訂第7版

ウェブアプリケーションのセキュリティ実装と
ウェブサイトの安全性向上のための取り組み



IPA

独立行政法人 情報処理推進機構
セキュリティセンター

2021年3月

本書は、以下の URL からダウンロードできます。

「安全なウェブサイトの作り方」

<https://www.ipa.go.jp/security/vuln/websecurity.html>

目次

目次	1
はじめに	2
本書の内容および位置付け	3
対象読者	3
第7版の主な改訂内容	3
脆弱性対策について ―根本的解決と保険的対策―	4
1. ウェブアプリケーションのセキュリティ実装	5
1.1 SQL インジェクション	6
1.2 OS コマンド・インジェクション	10
1.3 パス名パラメータの未チェック/ディレクトリ・トラバーサル	13
1.4 セッション管理の不備	16
1.5 クロスサイト・スクリプティング	22
1.6 CSRF(クロスサイト・リクエスト・フォージェリ)	30
1.7 HTTP ヘッダ・インジェクション	34
1.8 メールヘッダ・インジェクション	38
1.9 クリックジャッキング	41
1.10 バッファオーバーフロー	44
1.11 アクセス制御や認可制御の欠落	46
2. ウェブサイトの安全性向上のための取り組み	48
2.1 ウェブサーバに関する対策	48
2.2 DNS に関する対策	49
2.3 ネットワーク盗聴への対策	51
2.4 フィッシング詐欺を助長しないための対策	53
2.5 パスワードに関する対策	55
2.6 WAF によるウェブアプリケーションの保護	58
2.7 携帯ウェブ向けのサイトにおける注意点	64
3. 失敗例	71
3.1 SQL インジェクションの例	71
3.2 OS コマンド・インジェクションの例	77
3.3 パス名パラメータの未チェックの例	79
3.4 不適切なセッション管理の例	81
3.5 クロスサイト・スクリプティングの例	84
3.6 CSRF(クロスサイト・リクエスト・フォージェリ)の例	95
3.7 HTTP ヘッダ・インジェクションの例	99
3.8 メールヘッダ・インジェクションの例	100
おわりに	103
用語集	104
チェックリスト	105
CWE 対応表	109
更新履歴	111

はじめに

インターネットでは、多くのウェブサイトがそれぞれサービスを提供しています。通信利用動向調査¹によると、2015 年現在、日本におけるインターネットの利用者数は 1 億人を超えると推定され、ウェブを通じた情報のやり取りは今後も増え続けることが予想されます。

一方、ウェブサイトの「安全上の欠陥」(脆弱性)が狙われる事件も後を絶ちません。最近では営利目的の犯行も目立ち、悪質化が進む傾向にあります。独立行政法人 情報処理推進機構 (IPA) が届出²を受けたウェブサイトの脆弱性関連情報は、届出受付開始から 2020 年 12 月末時点で累計 11,526 件となりました。中でも「SQL インジェクション」と呼ばれる脆弱性は、ウェブサイトから個人情報などを不正に盗まれたり、ウェブページにウイルスを埋め込まれたりするといった事件における原因の一つと考えられています。

ウェブサイトの安全を維持するためには、ウェブサイトを構成する要素に対して、それぞれに適した対策を実施する必要があります。たとえば、サーバ OS やソフトウェアに対しては、各ベンダが提供する情報を元に、脆弱性修正パッチの適用や安全な設定等、共通した対応を実施することができます。しかし、「ウェブアプリケーション」については、それぞれのウェブサイトで独自に開発するケースが多く、セキュリティ対策はそれぞれのウェブアプリケーションに対して個別に実施する必要があります。すでに運用を開始しているウェブアプリケーションにセキュリティ上の問題が発覚した場合、設計レベルから修正することは難しい場合が少なくなく、場あたりの対策で済まざるをえないこともあります。対策は可能な限り、根本的な解決策を開発段階で実装することが望まれます。

本書は、IPA が届出を受けたソフトウェア製品およびウェブアプリケーションの脆弱性関連情報に基づいて、特にウェブサイトやウェブアプリケーションについて、届出件数の多かった脆弱性や攻撃による影響度が大きい脆弱性を取り上げ、その根本的な解決策と、保険的な対策を示しています。また、ウェブサイト全体の安全性を向上するための取り組みや、ウェブアプリケーション開発者が陥りやすい失敗例を紹介しています。

本書が、ウェブサイトのセキュリティ問題を解決する一助となれば幸いです。

¹ 総務省：通信利用動向調査

<https://www.soumu.go.jp/johotsusintokei/statistics/statistics05.html>

² IPA セキュリティセンターでは、経済産業省の告示に基づき、脆弱性情報に関する届出を受け付けています。脆弱性関連情報の届出

<https://www.ipa.go.jp/security/vuln/report/index.html>

本書の内容および位置付け

本書は、脆弱性関連情報流通の基本枠組みである「情報セキュリティ早期警戒パートナーシップ」の「受付・分析機関」である IPA において、実際に脆弱性と判断している問題を主に取り上げています。

本書は 3 章で構成しています。

第 1 章では、「ウェブアプリケーションのセキュリティ実装」として、SQL インジェクション、OS コマンド・インジェクションやクロスサイト・スクリプティング等 11 種類の脆弱性を取り上げ、それぞれの脆弱性で発生しうる脅威や特に注意が必要なウェブサイトの特徴等を解説し、脆弱性の原因そのものをなくす根本的な解決策、攻撃による影響の低減を期待できる対策を示しています。

第 2 章では、「ウェブサイトの安全性向上のための取り組み」として、ウェブサーバのセキュリティ対策やフィッシング詐欺を助長しないための対策等 7 つの項目を取り上げ、主に運用面からウェブサイト全体の安全性を向上させるための方策を示しています。

第 3 章では、「失敗例」として、第 1 章で取り上げた脆弱性の中から 8 種類を取り上げ、ウェブアプリケーションに脆弱性を作り込んでしまった際のソースコード、その解説、修正例を示しています。

巻末には、ウェブアプリケーションのセキュリティ実装の実施状況を確認するためのチェックリストや、CWE 対応表も付属しています。

本書に示す内容は、あくまで解決策の一例であり、必ずしもこれらの実施を求めるものではありません。また、修正例として紹介しているソースコードは、簡易検証によりその有効性を確認していますが、副作用が無いことを保証するものではありません。ウェブサイトのセキュリティ問題の解決の参考にしていただければ幸いです。

対象読者

対象読者は、企業や個人を問わず、ウェブアプリケーション開発者やサーバ管理者等、ウェブサイトの運営に関わる方の全てとしています。特に、セキュリティを初めて意識するウェブアプリケーション開発者の方を想定しています。

第 7 版の主な改訂内容

第 7 版では、1 章に、クリックジャッキングとバッファオーバーフローの脆弱性の解説を追加し、クロスサイト・スクリプティングの脆弱性への対策方法、各脆弱性で紹介している届出状況、参考 URL 等を更新しました。また、2 章に、ウェブサイトにおけるパスワードの管理方法の解説を追加し、通信経路の暗号化の解説、DNS などの対策方法、参考 URL を更新しました。

脆弱性対策について — 根本的解決と保険的対策 —

脆弱性への対策は、その対策内容や取り組みの視点によって、期待できる効果が異なります。ある対策は、脆弱性の原因そのものを取り除く、根本からの解決を期待できるものかもしれません。また、ある対策は、外因である攻撃手法に着目して特定の攻撃は防ぐことができるものの、別の種類の攻撃に対しては効果がないものかもしれません。ここで大切なことは、自分が選択する対策が、どのような性質を持っているのか、期待する効果を得られるものなのか、ということのを正しく理解、把握することです。

本書では、特にウェブアプリケーションにおける脆弱性対策について、その性質を基に「根本的解決」と「保険的対策」の2つに分類しています。

■ 根本的解決

本書における「根本的解決」は、「脆弱性を作り込まない実装」を実現する手法です。根本的解決を実施することにより、その脆弱性を狙った攻撃が無効化されることを期待できます。

■ 保険的対策

本書における「保険的対策」は、「攻撃による影響を軽減する対策」です。根本的解決とは違って、脆弱性の原因そのものを無くすものではありませんが、攻撃から被害までの次の各フェーズにおいて、それぞれの影響を軽減できます。

- 攻撃される可能性を低減する
(例: 攻撃につながるヒントを与えない)
- 攻撃された場合に、脆弱性を突かれる可能性を低減する
(例: 入力から攻撃に使われるデータをサニタイズ(無効化)する)
- 脆弱性を突かれた場合に、被害範囲を最小化する
(例: アクセス制御)
- 被害が生じた場合に、早期に知る
(例: 事後通知)

理想的には、ウェブアプリケーション開発の設計段階から、根本的解決の手法を採用することが望ましいと言えます。保険的対策は脆弱性の原因そのものを無くす対策ではありませんので、保険的対策のみに頼る設計は推奨されません。とはいえ、根本的解決の実装に漏れが生じる場合、保険的対策はいわば「セーフティネット」として機能しますので、根本的解決と保険的対策を併せて採用することが有効な場合もあります。

すでに開発を終え運用段階のウェブアプリケーションにおいて、後から脆弱性対策を実施する場合においても、根本的解決の手法を採用することが望ましいですが、費用や時間、その他の事情によりすぐに実施できない場合には、保険的対策は暫定対策として機能します。

保険的対策は、対策の内容によっては、本来の機能を制限することになるものもあるので、そのような副作用の影響も考慮する必要があります。

1. ウェブアプリケーションのセキュリティ実装

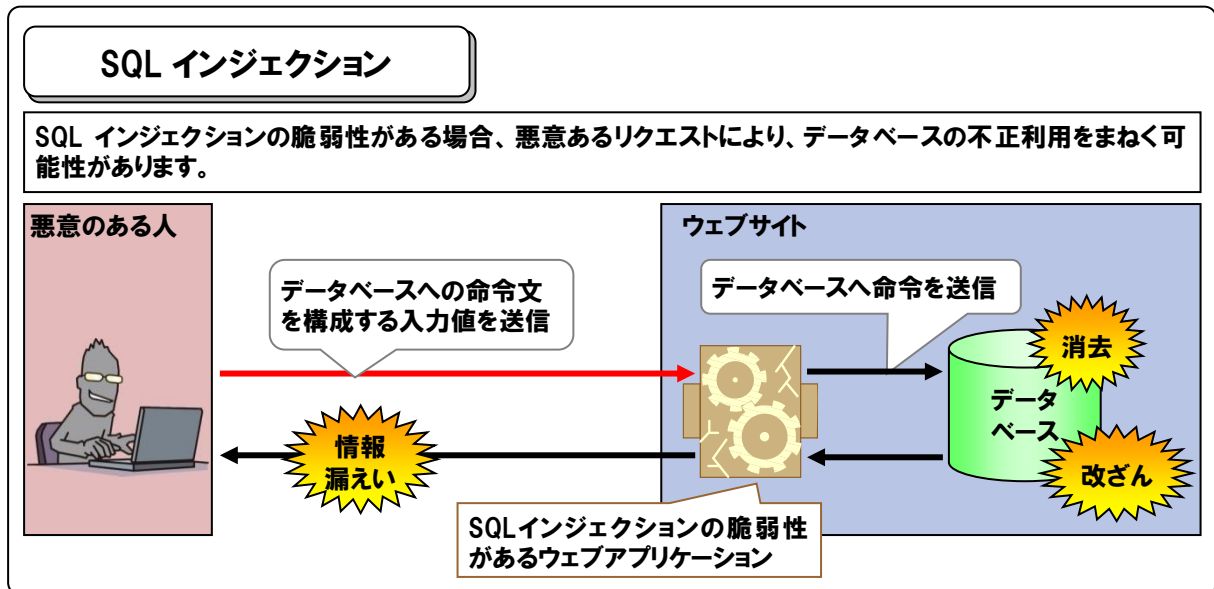
本章では、ウェブアプリケーションのセキュリティ実装として、下記の脆弱性を取り上げ³、発生しうる脅威、注意が必要なサイト、根本的解決および保険的対策を示します。

- 1) SQL インジェクション
- 2) OS コマンド・インジェクション
- 3) パス名パラメータの未チェック／ディレクトリ・トラバーサル
- 4) セッション管理の不備
- 5) クロスサイト・スクリプティング
- 6) CSRF (クロスサイト・リクエスト・フォージェリ)
- 7) HTTP ヘッダ・インジェクション
- 8) メールヘッダ・インジェクション
- 9) クリックジャッキング
- 10) バッファオーバーフロー
- 11) アクセス制御や認可制御の欠落

³ 資料の構成上、脆弱性の深刻度や攻撃による影響、届出状況を考慮して項番を割り当てていますが、これは対策の優先順位を示すものではありません。優先順位は運営するウェブサイトの状況に合わせてご検討ください。

1.1 SQL インジェクション

データベースと連携したウェブアプリケーションの多くは、利用者からの入力情報を基に SQL 文(データベースへの命令文)を組み立てています。ここで、SQL 文の組み立て方法に問題がある場合、攻撃によってデータベースの不正利用をまねく可能性があります。このような問題を「SQL インジェクションの脆弱性」と呼び、問題を悪用した攻撃を、「SQL インジェクション攻撃」と呼びます。



■ 発生しうる脅威

SQL インジェクション攻撃により、発生しうる脅威は次のとおりです。

- データベースに蓄積された非公開情報の閲覧
 - 個人情報 の漏えい 等
- データベースに蓄積された情報の改ざん、消去
 - ウェブページ の改ざん、パスワード変更、システム停止 等
- 認証回避による不正ログイン⁴
 - ログインした利用者 に許可されている全ての操作を不正に行われる
- ストアドプロシージャ等を利用した OS コマンドの実行
 - システムの乗っ取り、他への攻撃の踏み台としての悪用 等

■ 注意が必要なウェブサイトの特徴

運営主体やウェブサイトの性質を問わず、データベース⁵ を利用するウェブアプリケーションを設置しているウェブサイトが存在しうる問題です。個人情報等の重要情報をデータベースに格納しているウェブサイトは、特に注意が必要です。

⁴ 後述「1.4 セッション管理の不備」で解説する「発生しうる脅威」と同じ内容です。

⁵ 代表的なデータベースエンジンには、MySQL, PostgreSQL, Oracle, Microsoft SQL Server, DB2 等が挙げられます。

■ 届出状況⁶

SQL インジェクションの脆弱性に関する届出件数は他の脆弱性に比べて多く、届出受付開始から2014年第4四半期までに、ウェブサイトの届出件数の11%に相当する届出を受けています。また、ソフトウェア製品の届出も、ウェブサイトの届出件数ほど多くはありませんが、少なからずあります。下記は、IPAが届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

・「DBD::PgPP」における SQL インジェクションの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDDB-2014-000142>

・「Piwigo」における SQL インジェクションの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDDB-2014-000094>

・「サイボウズ ガルーン」における SQL インジェクションの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDDB-2014-000024>

■ 根本的解決

1-(i)-a

👉 **SQL 文の組み立ては全てプレースホルダで実装する。**

SQL には通常、プレースホルダを用いて SQL 文を組み立てる仕組みがあります。SQL 文の雛形の中に変数の場所を示す記号(プレースホルダ)を置いて、後に、そこに実際の値を機械的な処理で割り当てるものです。ウェブアプリケーションで直接、文字列連結処理によって SQL 文を組み立てる方法に比べて、プレースホルダでは、機械的な処理で SQL 文が組み立てられるので、SQL インジェクションの脆弱性を解消できます。

プレースホルダに実際の値を割り当てる処理をバインドと呼びます。バインドの方式には、プレースホルダのまま SQL 文をコンパイルしておき、データベースエンジン側で値を割り当てる方式(静的プレースホルダ)と、アプリケーション側のデータベース接続ライブラリ内で値をエスケープ処理してプレースホルダにはめ込む方式(動的プレースホルダ)があります。静的プレースホルダは、SQL の ISO/JIS 規格では、準備された文(Prepared Statement)と呼ばれます。

どちらを用いても SQL インジェクション脆弱性を解消できますが、原理的に SQL インジェクション脆弱性の可能性がなくなるという点で、静的プレースホルダの方が優れます。詳しくは本書別冊の「安全な SQL の呼び出し方」のプレースホルダの項(3.2 節)を参照してください。

1-(i)-b

👉 **SQL 文の組み立てを文字列連結により行う場合は、エスケープ処理等を行うデータベースエンジンの API を用いて、SQL 文のリテラルを正しく構成する。**

SQL 文の組み立てを文字列連結により行う場合は、SQL 文中で可変となる値をリテラル(定数)の形で埋め込みます。値を文字列型として埋め込む場合は、値をシングルクォートで囲んで記述しますが、その際に文字列リテラル内で特別な意味を持つ記号文字をエスケープ処理します(たとえば、「'」→「''」、「¥」→「¥¥」等)。値を数値型として埋め込む場合は、数値リテラルであることを確実にする処

⁶ 最新情報は、下記 URL を参照してください。

脆弱性関連情報に関する届出状況: <https://www.ipa.go.jp/security/vuln/report/press.html>

理(数値型へのキャスト等)を行います。

こうした処理で具体的に何をすべきかは、データベースエンジンの種類や設定によって異なるため、それに合わせた実装が必要です。データベースエンジンによっては、リテラルを文字列として生成する専用の API⁷を提供しているものがありますので、それを利用することをお勧めします。詳しくは、「安全な SQL の呼び出し方」の 4.1 節を参照してください。

なお、この処理は、外部からの入力の影響を受ける値のみに限定して行うのではなく、SQL 文を構成する全てのリテラル生成に対して行うべきです。

1-(ii)

👉 **ウェブアプリケーションに渡されるパラメータに SQL 文を直接指定しない。**

これは、いわば「論外」の実装ですが、hidden パラメータ等に SQL 文をそのまま指定するという事例の届出がありましたので、避けるべき実装として紹介します。

ウェブアプリケーションに渡されるパラメータに SQL 文を直接指定する実装は、そのパラメータ値の変更により、データベースの不正利用につながる可能性があります。

■ 保険的対策

1-(iii)

👉 **エラーメッセージをそのままブラウザに表示しない。**

エラーメッセージの内容に、データベースの種類やエラーの原因、実行エラーを起こした SQL 文等の情報が含まれる場合、これらは SQL インジェクション攻撃につながる有用な情報となりえます。また、エラーメッセージは、攻撃の手がかりを与えるだけでなく、実際に攻撃された結果を表示する情報源として悪用される場合があります。データベースに関連するエラーメッセージは、利用者のブラウザ上に表示させないことをお勧めします。

1-(iv)

👉 **データベースアカウントに適切な権限を与える。**

ウェブアプリケーションがデータベースに接続する際に使用するアカウントの権限が必要以上に高い場合、攻撃による被害が深刻化する恐れがあります。ウェブアプリケーションからデータベースに渡す命令文を洗い出し、その命令文の実行に必要な最小限の権限をデータベースアカウントに与えてください。

以上の対策により、SQL インジェクション攻撃に対する安全性の向上が期待できます。データベースと連携したウェブアプリケーションの構築や、SQL インジェクションの脆弱性に関する情報については、次の資料も参考にしてください。

⁷ 実行環境によっては、エスケープ処理を適切に行わない脆弱性が指摘されている API もあります。その場合は修正パッチを適用するか、別の方法を検討して下さい。

■ CWE

CWE-89: SQL インジェクション

<https://jvndb.jvn.jp/ja/cwe/CWE-89.html>

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (2.8)

<https://cwe.mitre.org/data/definitions/89.html>

■ 参考 URL

IPA: 安全な SQL の呼び出し方

<https://www.ipa.go.jp/security/vuln/websecurity.html#sql>

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「1. SQL インジェクション」

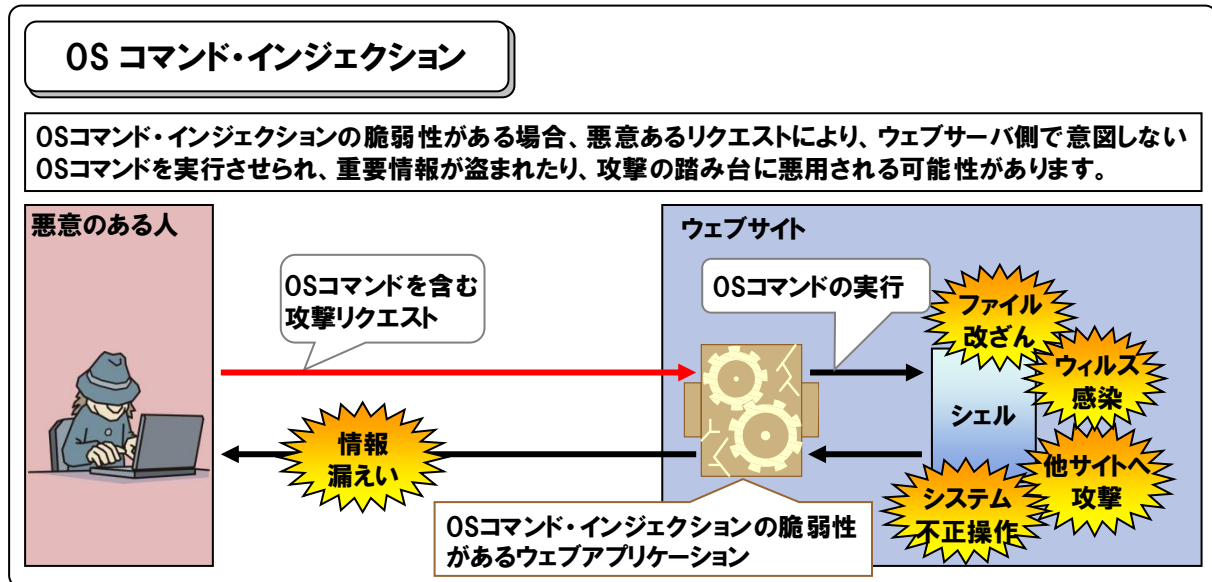
https://www.ipa.go.jp/security/vuln/vuln_contents/sql.html

IPA: 「2014 年度情報セキュリティ事象被害状況調査」報告書について

<https://www.ipa.go.jp/security/fy26/reports/isec-survey/index.html>

1.2 OS コマンド・インジェクション

ウェブアプリケーションによっては、外部からの攻撃により、ウェブサーバの OS コマンドを不正に実行されてしまう問題を持つものがあります。このような問題を「OS コマンド・インジェクションの脆弱性」と呼び、問題を悪用した攻撃手法を、「OS コマンド・インジェクション攻撃」と呼びます。



■ 発生しうる脅威

OS コマンド・インジェクション攻撃により、発生しうる脅威は次のとおりです。

- サーバ内ファイルの閲覧、改ざん、削除

重要情報の漏えい、設定ファイルの改ざん 等

- 不正なシステム操作

意図しない OS のシャットダウン、ユーザアカウントの追加、変更 等

- 不正なプログラムのダウンロード、実行

ウイルス、ワーム、ボット等への感染、バックドアの設置 等

- 他のシステムへの攻撃の踏み台

サービス不能攻撃、システム攻略のための調査、迷惑メールの送信 等

■ 注意が必要なウェブサイトの特徴

運営主体やウェブサイトの性質を問わず、外部プログラムを呼び出し可能な関数等⁸を使用しているウェブアプリケーションに注意が必要な問題です。

⁸ 外部プログラムを呼び出し可能な関数の例：

Perl : open(), system(), eval() 等

PHP : exec(), passthru(), shell_exec(), system(), popen() 等

■ 届出状況

OS コマンド・インジェクションの脆弱性は、Perl で開発されたウェブアプリケーションや、組み込み製品の管理画面で使用される CGI プログラム等のソフトウェア製品に発見され、届出を受けています。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・複数の ASUS 製無線 LAN ルータにおける OS コマンド・インジェクションの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDDB-2015-000011>

- ・「Usermin」における OS コマンド・インジェクションの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDDB-2014-000057>

- ・「Movable Type」における OS コマンド・インジェクションの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDDB-2012-000017>

■ 根本的解決

2-(i)

☞ シェルを起動できる言語機能の利用を避ける。

ウェブアプリケーションに利用されている言語によっては、シェルを起動できる機能を持つものがあります。たとえば、Perl の open 関数等です。Perl の open 関数は、引数として与えるファイルパスに「|」（パイプ）を使うことで OS コマンドを実行できるため、外部からの入力値を引数として利用する実装は危険です。シェルを起動できる言語機能の利用は避けて⁹、他の関数等で代替してください。Perl でファイルを開く場合、sysopen 関数を利用すればシェルを起動することはありません。

■ 保険的対策

2-(ii)

☞ シェルを起動できる言語機能を利用する場合は、その引数を構成する全ての変数に対してチェックを行い、あらかじめ許可した処理のみを実行する。

シェルを起動できる言語機能の引数を構成する変数に対し、引数に埋め込む前にチェックをかけ、本来想定する動作のみを実行するように実装してください。チェック方法には、その引数に許可する文字の組み合わせを洗い出し、その組み合わせ以外は許可しない「ホワイトリスト方式」をお勧めします。数値を示すはずのパラメータであれば、数字のみからなる文字列であることをチェックします。チェックの結果、許可しない文字の組み合わせが確認された場合は、引数へ渡さず、処理を中止します。

なお、チェック方法には、OS コマンド・インジェクション攻撃に悪用される記号文字（「|」、「<」、「>」等）等、問題となりうる文字を洗い出し、これを許可しない「ブラックリスト方式」もありますが、この方法はチェックに漏れが生じる可能性があるため、お勧めできません。

以上の対策により、OS コマンド・インジェクション攻撃に対する安全性の向上が期待できます。OS コマンド・インジェクションの脆弱性に関する情報については、次の資料も参考にしてください。

⁹ 3.2 の修正例 1~3 を参照。

■ CWE

CWE-78: OS コマンドインジェクション

<https://jvndb.jvn.jp/ja/cwe/CWE-78.html>

CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

<https://cwe.mitre.org/data/definitions/78.html>

■ 参考 URL

IPA: 知っていますか?脆弱性(ぜいじゃくせい)「5. OS コマンド・インジェクション」

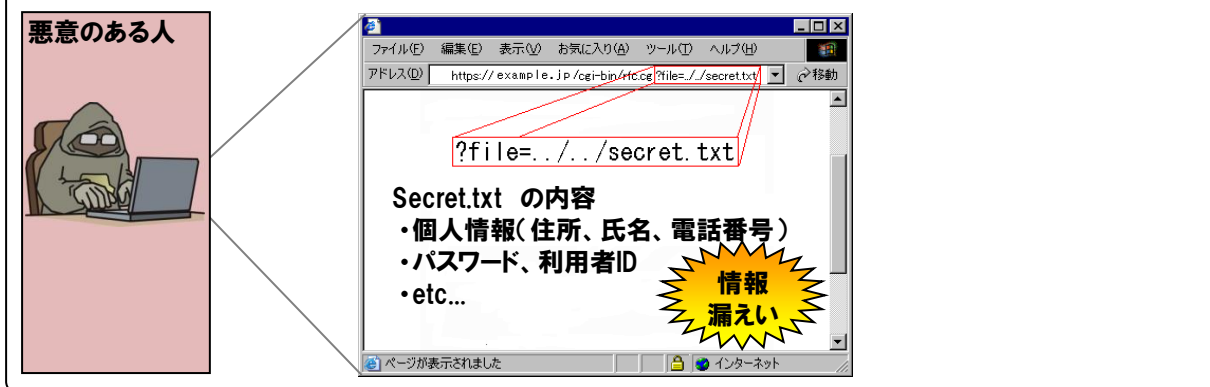
https://www.ipa.go.jp/security/vuln/vuln_contents/oscmd.html

1.3 パス名パラメータの未チェック/ディレクトリ・トラバーサル

ウェブアプリケーションの中には、外部からのパラメータにウェブサーバ内のファイル名を直接指定しているものがあります。このようなウェブアプリケーションでは、ファイル名指定の実装に問題がある場合、攻撃者に任意のファイルを指定され、ウェブアプリケーションが意図しない処理を行ってしまう可能性があります。このような問題の一種を「ディレクトリ・トラバーサルの脆弱性」と呼び、この問題を悪用した攻撃手法の一つに、「ディレクトリ・トラバーサル攻撃」があります。

パス名パラメータを悪用したファイル参照

パラメータにファイル名を指定しているウェブアプリケーションでは、ファイル名指定の実装に問題がある場合、公開を想定していないファイルを参照されてしまう可能性があります。



■ 発生しうる脅威

本脆弱性を悪用した攻撃により、発生しうる脅威は次のとおりです。

- サーバ内ファイルの閲覧、改ざん、削除

- ・ 重要情報の漏えい
- ・ 設定ファイル、データファイル、プログラムのソースコード等の改ざん、削除

■ 注意が必要なウェブサイトの特徴

運営主体やウェブサイトの性質を問わず、外部からのパラメータにウェブサーバ内のファイル名を直接指定しているウェブアプリケーションに起こりうる問題です。個人情報等の重要情報をウェブサーバ内にファイルとして保存しているサイトは、特に注意が必要です。

- サーバ内ファイルを利用するウェブアプリケーションの例

- ・ ウェブページのデザインテンプレートをファイルから読み込む
- ・ 利用者からの入力内容を指定のファイルへ書き込む 等

■ 届出状況

パス名パラメータに関する脆弱性の届出がウェブサイトの届出全体に占める割合は、数パーセントと多くはありません。しかしながら、これらの脆弱性については受付開始当初から継続して届出を受けてい

ます。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・シンクグラフィカ製「ダウンロードログ CGI」におけるディレクトリ・トラバーサルの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDB-2015-000006>

- ・「Spring Framework」におけるディレクトリ・トラバーサルの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDB-2014-000054>

- ・「VMware ESX および ESXi」におけるディレクトリ・トラバーサルの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDB-2013-000084>

■ 根本的解決

3-(i)-a

☞ **外部からのパラメータでウェブサーバ内のファイル名を直接指定する実装を避ける。**

外部からのパラメータでウェブサーバ内のファイル名を直接指定する実装では、そのパラメータが変更され、任意のファイル名を指定されることにより公開を想定しないファイルが外部から閲覧される可能性があります。たとえば、HTML 中の hidden パラメータでウェブサーバ内のファイル名を指定し、そのファイルをウェブページのテンプレートとして使用する実装では、そのパラメータが変更されることで、任意のファイルをウェブページとして出力してしまう等の可能性があげられます。

外部からのパラメータでウェブサーバ内のファイル名を直接指定する実装が本当に必要か、他の処理方法で代替できないか等、仕様や設計から見直すことをお勧めします。

3-(i)-b

☞ **ファイルを開く際は、固定のディレクトリを指定し、かつファイル名にディレクトリ名が含まれないようにする。**

たとえば、カレントディレクトリ上のファイル「filename」を開くつもりで、`open(filename)` の形式でコーディングしている場合、`open(filename)` の filename に絶対パス名が渡されることにより、任意ディレクトリのファイルが開いてしまう可能性があります。この絶対パス名による指定を回避する方法として、あらかじめ固定のディレクトリ「dirname」を指定し、`open(dirname+filename)` のような形でコーディングする方法があります。しかし、これだけでは、「./」等を使用したディレクトリ・トラバーサル攻撃を回避できません。これを回避するために、`basename()` 等の、パス名からファイル名のみを取り出す API を利用して、`open(dirname+basename(filename))` のような形でコーディングして、filename に与えられたパス名からディレクトリ名を取り除くようにします¹⁰。

■ 保険的対策

3-(ii)

☞ **ウェブサーバ内のファイルへのアクセス権限の設定を正しく管理する。**

ウェブサーバ内に保管しているファイルへのアクセス権限が正しく管理されていれば、ウェブアプリ

¹⁰ 3.3 の修正例を参照。

ケーションが任意ディレクトリのファイルを開く処理を実行しようとしても、ウェブサーバ側の機能でそのアクセスを拒否できる場合があります。

3-(iii)

👁️ ファイル名のチェックを行う。

ファイル名を指定した入力パラメータの値から、「/」、「../」、「..¥」等、OS のパス名解釈でディレクトリを指定できる文字列を検出した場合は、処理を中止します。ただし、URL のデコード処理を行っている場合は、URL エンコードした「%2F」、「..%2F」、「..%5C」、さらに二重エンコードした「%252F」、「..%252F」、「..%255C」がファイル指定の入力値として有効な文字列となる場合があります。チェックを行うタイミングに注意してください。

以上の対策により、パス名パラメータを悪用した攻撃に対する安全性の向上が期待できます。本脆弱性に関する情報については、次の資料も参考にしてください。

■ CWE

CWE-22: パス・トラバーサル

<https://jvndb.jvn.jp/ja/cwe/CWE-22.html>

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

<https://cwe.mitre.org/data/definitions/22.html>

■ 参考 URL

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「4. パス名パラメータの未チェック／ディレクトリ・トラバーサル」

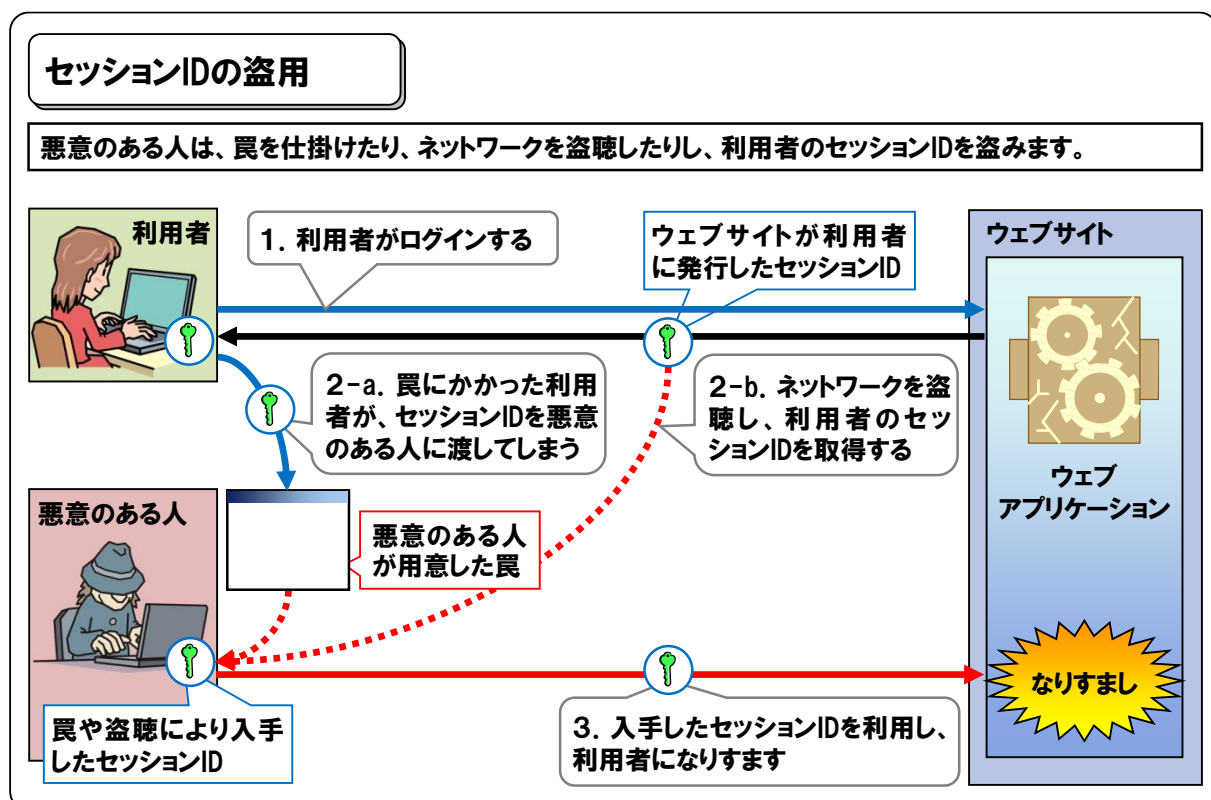
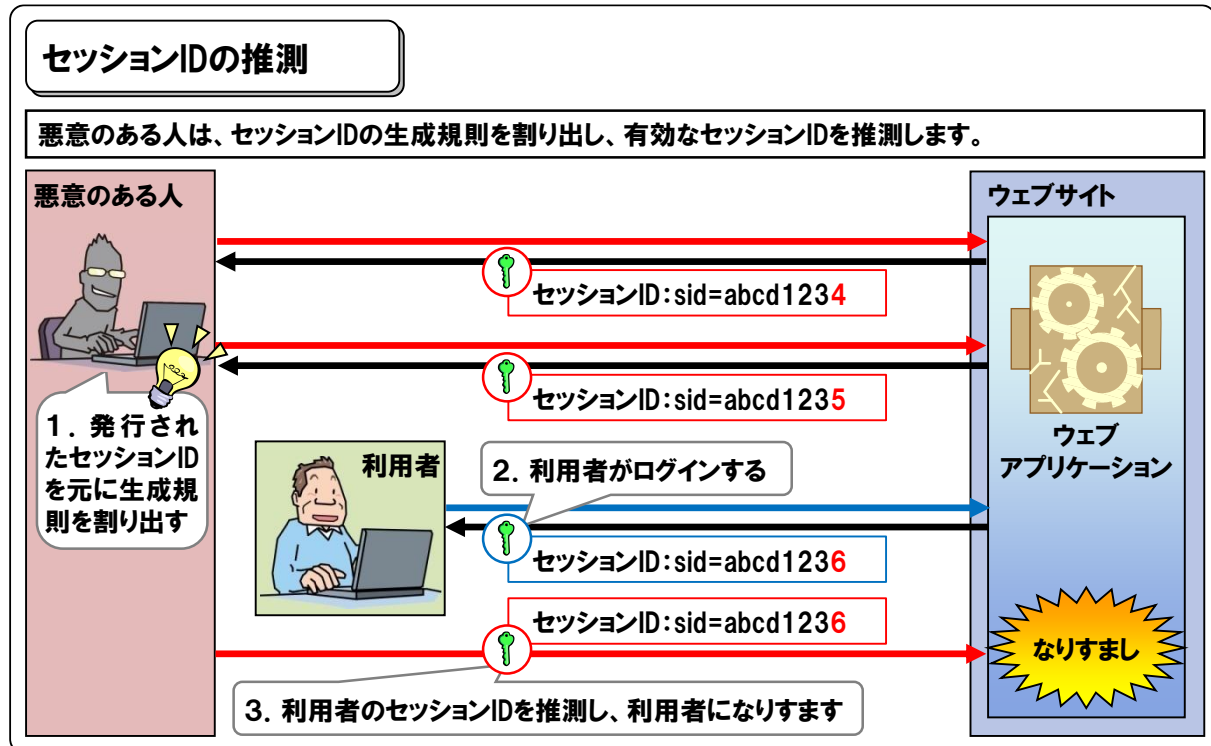
https://www.ipa.go.jp/security/vuln/vuln_contents/dt.html

IPA: ソフトウェア等の脆弱性関連情報に関する届出状況[2014 年第 4 四半期(10 月～12 月)] 1-4.節

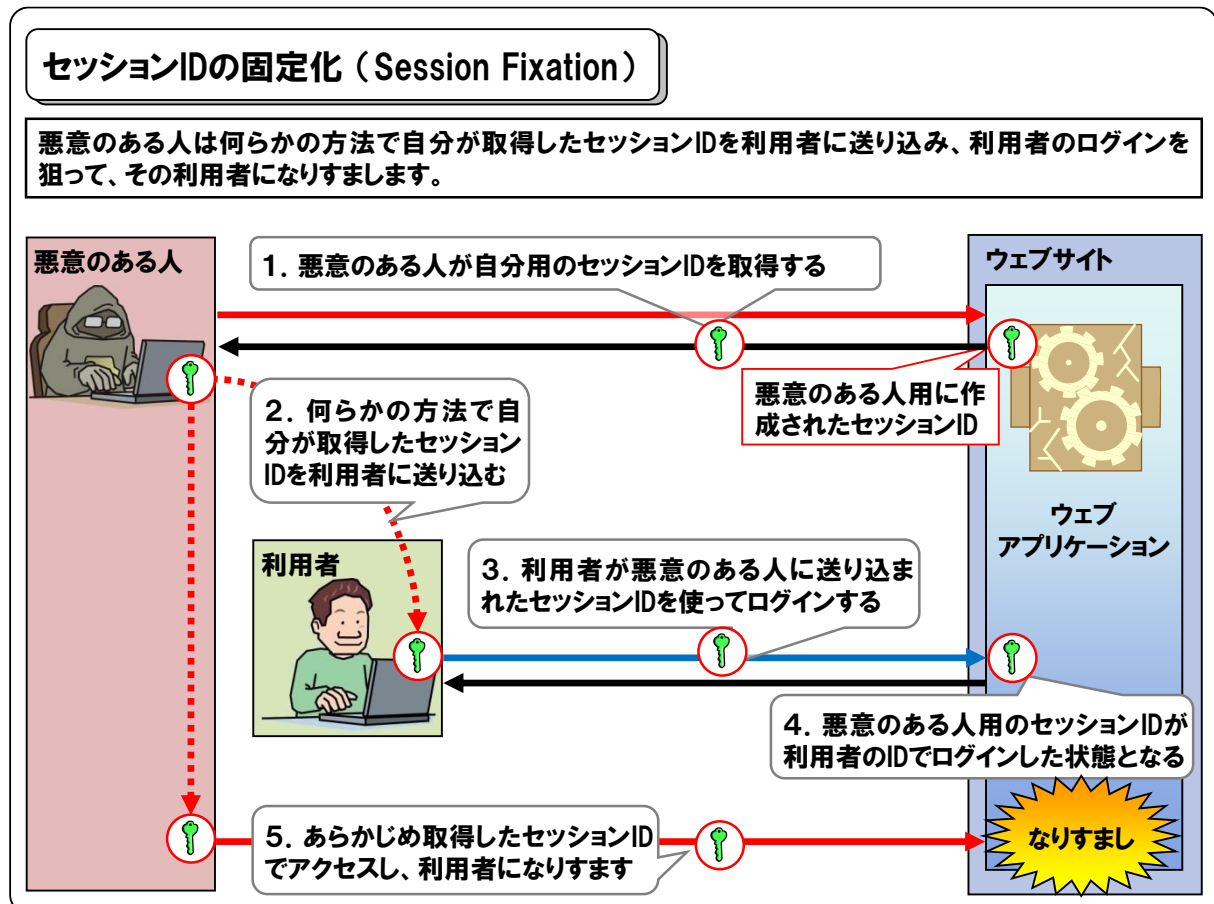
<https://www.ipa.go.jp/security/vuln/report/vuln2014q4.html#t04>

1.4 セッション管理の不備

ウェブアプリケーションの中には、セッション ID (利用者を識別するための情報) を発行し、セッション管理を行っているものがあります。このセッション ID の発行や管理に不備がある場合、悪意のある人にログイン中の利用者のセッション ID を不正に取得され、その利用者になりすましてアクセスされてしまう可能性があります。この問題を悪用した攻撃手法を、「セッション・ハイジャック」と呼びます。



また、推測や盗用以外に、セッション管理の不備を狙ったもう一つの攻撃手法として、「セッション ID の固定化 (Session Fixation)」と呼ばれる攻撃手法があります。悪意ある人があらかじめ用意したセッション ID を、何らかの方法¹¹で利用者に送り込み、利用者がこれに気付かずにパスワードを入力するなどしてログインすると起こりうる問題です。悪意のある人がこの攻撃に成功すると、あらかじめ用意したセッション ID を利用し、利用者になりすましてウェブサイトへアクセスすることができてしまいます。



■ 発生しうる脅威

セッション管理の不備を狙った攻撃が成功した場合、攻撃者は利用者になりすまし、その利用者本人に許可されている操作を不正に行う可能性があります。具体的には、次の脅威が発生します。

¹¹ 用意したセッション ID を利用者に送り込むことができちゃうのは、次のいずれかに該当する場合です。

1. ウェブアプリケーションがセッション ID を POST メソッドの hidden パラメータに格納して受け渡している実装となっている場合
2. ウェブアプリケーションがセッション ID を Cookie に格納して受け渡している実装となっている場合で、利用者のウェブブラウザが、ドメインをまたがった Cookie のセットができちゃう「Cookie Monster(※1)」と呼ばれる問題を抱えている場合
3. ウェブアプリケーションがセッション ID を Cookie に格納して受け渡している実装となっている場合で、ウェブアプリケーションサーバ製品に、「Session Adoption(※2)」の脆弱性がある場合
4. ウェブアプリケーションにクロスサイト・スクリプティング(後述 1.5 参照)等他の脆弱性がある場合

※1 「Multiple Browser Cookie Injection Vulnerabilities」 <http://www.westpoint.ltd.uk/advisories/wp-04-0001.txt>

※2 「Session Fixation Vulnerability in Web-based Applications」 http://www.acrossecurity.com/papers/session_fixation.pdf

- ログイン後の利用者のみが利用可能なサービスの悪用

不正な送金、利用者が意図しない商品購入、利用者が意図しない退会処理 等

- ログイン後の利用者のみが編集可能な情報の改ざん、新規登録

各種設定の不正な変更(管理者画面、パスワード等)、掲示板への不適切な書き込み 等

- ログイン後の利用者のみが閲覧可能な情報の閲覧

非公開の個人情報を不正閲覧、ウェブメールを不正閲覧、コミュニティ会員専用の掲示板を不正閲覧 等

■ 注意が必要なウェブサイトの特徴

運営主体やウェブサイトの性質を問わず、ログイン機能を持つウェブサイト全般に注意が必要な問題です。ログイン後に決済処理等の重要な処理を行うサイトは、攻撃による被害が大きくなるため、特に注意が必要です。

- 金銭処理が発生するサイト

ネットバンキング、ネット証券、ショッピング、オークション 等

- 非公開情報を扱うサイト

転職サイト、コミュニティサイト、ウェブメール 等

- その他、ログイン機能を持つサイト

管理者画面、会員専用サイト、日記サイト 等

■ 届出状況

セッション管理の不備に関する届出がウェブサイトの届出全体に占める割合は、数パーセントと多くはありません。しかしながら、これらの脆弱性については受付開始当初から継続して届出を受けています。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

・「WisePoint」におけるセッション固定の脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2014-000084>

・「HDL-A」および「HDL2-A」シリーズにおけるセッション管理に関する脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2013-000095>

・「baserCMS」におけるセッション管理不備の脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2012-000043>

■ 根本的解決

4-(i)

👁 セッション ID を推測が困難なものにする。

セッション ID が時刻情報等を基に単純なアルゴリズムで生成されている場合、その値は第三者に容

易に予測されてしまいます¹²。利用者がログインするタイミングで発行されるセッション ID の値を悪意ある人によって推測されると、悪意ある人がそのセッション ID を使って利用者になりすまし、本来は利用者しかアクセスできないウェブサイト等にアクセスできてしまいます。セッション ID は、生成アルゴリズムに暗号論的擬似乱数生成器を用いるなどして、予測困難なものにしてください。

セッション管理の仕組みが提供されるウェブアプリケーションサーバ製品を利用する場合は、その製品が提供するセッション管理の仕組みを利用している限り、自前でセッション ID を生成する必要はありません。自前でセッション管理の仕組みを構築しようとせず、そうしたウェブアプリケーション製品を利用することをお勧めします。

4-(ii)

☞ セッション ID を URL パラメータに格納しない。

セッション ID を URL パラメータに格納していると、利用者のブラウザが、Referer 送信機能によって、セッション ID の含まれた URL をリンク先のサイトへ送信してしまいます。悪意ある人にその URL を入手されると、セッション・ハイジャックされてしまいます。セッション ID は、Cookie に格納するか、POST メソッドの hidden パラメータに格納して受け渡しするようにしてください。

なお、ウェブアプリケーションサーバ製品によっては、利用者が Cookie の受け入れを拒否している場合、セッション ID を URL パラメータに格納する実装に自動的に切り替えてしまうものがあります。そのような機能は、製品の設定変更を行う等によって、自動切り替え機能を無効化することを検討してください。

4-(iii)

☞ HTTPS 通信で利用する Cookie には secure 属性を加える。

ウェブサイトが発行する Cookie には、secure 属性という設定項目があり、これが設定された Cookie は HTTPS 通信のみで利用されます。Cookie に secure 属性がない場合、HTTPS 通信で発行した Cookie は、経路が暗号化されていない HTTP 通信でも利用されるため、この HTTP 通信の盗聴により Cookie 情報を不正に取得されてしまう可能性があります。HTTPS 通信で利用する Cookie には secure 属性を必ず加えてください。かつ、HTTP 通信で Cookie を利用する場合は、HTTPS で発行する Cookie とは別のものを発行してください。

4-(iv)-a

☞ ログイン成功後に、新しくセッションを開始する。

ウェブアプリケーションによっては、ユーザがログインする前の段階(例えばサイトの閲覧を開始した時点)でセッション ID を発行してセッションを開始し、そのセッションをログイン後も継続して使用する実装のものがあります。しかしながら、この実装はセッション ID の固定化攻撃に対して脆弱な場合があります。このような実装を避け、ログインが成功した時点から新しいセッションを開始する(新しいセッション ID でセッション管理をする)ようにします。また、新しいセッションを開始する際には、既存のセッション

¹² 3.4 のよくある失敗例 1~2 を参照。

ン ID を無効化します¹³。こうすることにより、利用者が新しくログインしたセッションに対し、悪意のある人は事前に手に入れたセッション ID ではアクセスできなくなります。

4-(iv)-b

👉 ログイン成功後に、既存のセッション ID とは別に秘密情報を発行し、ページの遷移ごとにその値を確認する。

セッション ID とは別に、ログイン成功時に秘密情報を作成して Cookie にセットし、この秘密情報と Cookie の値が一致することを全てのページで確認する¹⁴ようにします。なお、この秘密情報の作成には、前述の根本的解決 4-(i) の「セッション ID を推測が困難なものにする」と同様の生成アルゴリズムや、暗号処理を用います。

ただし、次の場合には本対策は不要です。

- ・上記根本的解決 4-(iv)-a の実装方法を採用している場合
- ・セッション ID をログイン前には発行せず、ログイン成功後に発行する実装のウェブアプリケーションの場合

■ 保険的対策

4-(v)

👉 セッション ID を固定値にしない。

発行するセッション ID が利用者ごとに固定の値である場合、この情報が攻撃者に入手されると、時間の経過に関係なく、いつでも攻撃者からセッション・ハイジャックされてしまいます。セッション ID は、利用者のログインごとに新しく発行し、固定値にしないようにしてください。

4-(vi)

👉 セッション ID を Cookie にセットする場合、有効期限の設定に注意する。

Cookie は有効期限が過ぎるまでブラウザに保持されます。このため、ブラウザの脆弱性を悪用する等何らかの方法で Cookie を盗むことが可能な場合、その時点で保持されていた Cookie が盗まれる可能性があります。Cookie を発行する場合は、有効期限の設定に注意してください。

たとえば、Cookie の有効期限を短い日時に設定し、必要以上の期間、Cookie がブラウザに保存されないようにする等の対策をとります。

なお、Cookie をブラウザに残す必要が無い場合は、有効期限の設定(expires=)を省略し、発行した Cookie をブラウザ終了後に破棄させる方法もあります。しかし、この方法は、利用者がブラウザを終了させずに使い続けた場合には Cookie は破棄されないため、期待する効果を得られない可能性があります。

¹³ ログイン後にログイン前のセッション情報を引き継ぐ必要がある場合には、セッションデータのコピー方式に注意が必要です。オブジェクト変数を浅いコピー(shallow copy)で引き継いだ場合、ログイン前セッションとログイン後セッションが、同一のデータを共有して参照することになり、ログイン前のセッション ID によるアクセスで、ログイン後セッションのデータの一部を操作できてしまう危険性があります。また、ログイン後セッションのデータを、ログイン前のセッション ID によるアクセスで閲覧できてしまうことが、脆弱性となる場合も考えられます。これを防止するには、深いコピー(deep copy)で引き継ぐ方法も考えられますが、それだけではデータベースへの参照の共有や、一時ファイルへの参照の共有等が残り、脆弱性となる場合もあると考えられるので、ログイン成功時にログイン前のセッションを破棄する方法をお勧めします。

¹⁴ 一部のウェブアプリケーションサーバ製品では、このような処理を自動的に行う実装のものもあります。

ります。

以上の対策により、セッション・ハイジャック攻撃に対する安全性の向上が期待できます。セッション管理に関する情報については、次の資料も参考にしてください。

■ CWE

CWE-330: Use of Insufficiently Random Values

<https://cwe.mitre.org/data/definitions/330.html>

CWE-522: Insufficiently Protected Credentials

<https://cwe.mitre.org/data/definitions/522.html>

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

<https://cwe.mitre.org/data/definitions/614.html>

CWE-384: Session Fixation

<https://cwe.mitre.org/data/definitions/384.html>

■ 参考 URL

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「6. セッション管理の不備」

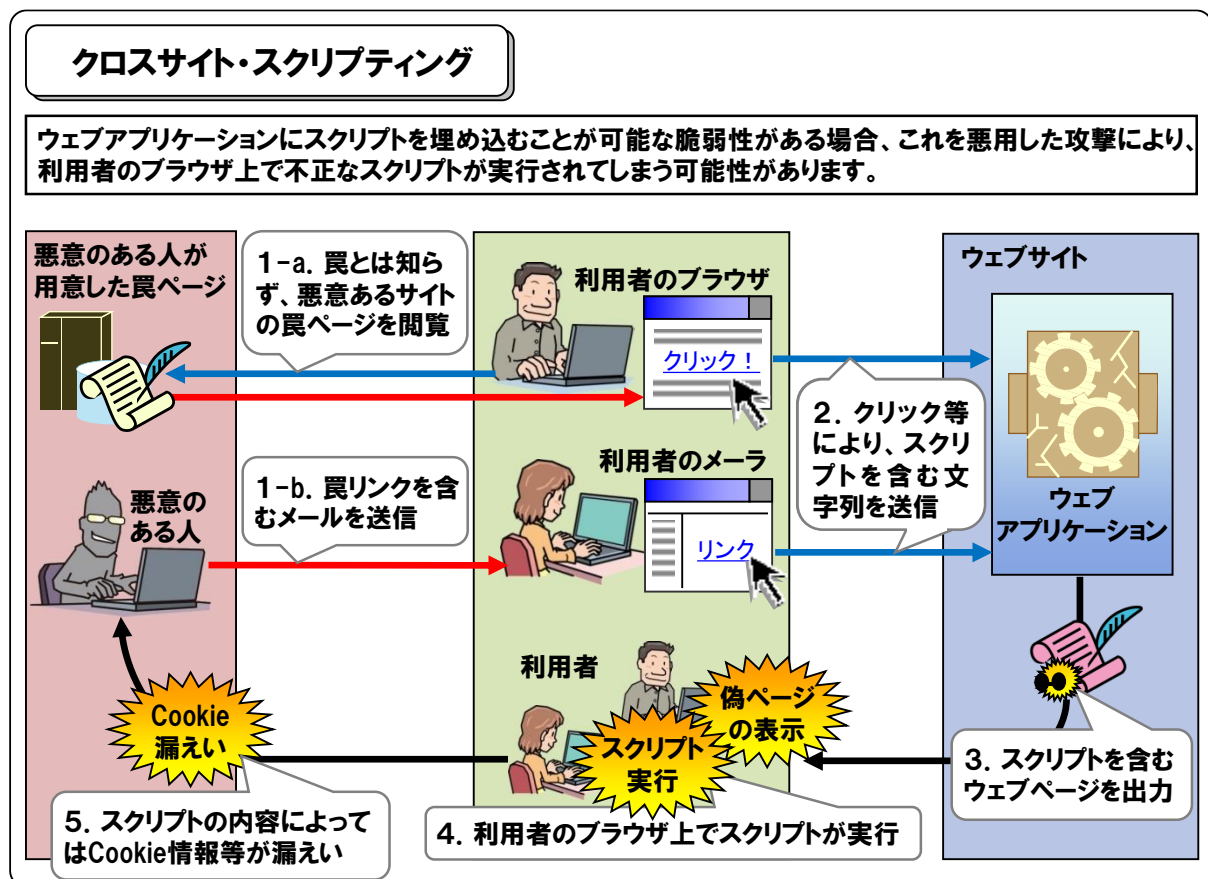
https://www.ipa.go.jp/security/vuln/vuln_contents/session.html

産業技術総合研究所 高木浩光: 「CSRF」と「Session Fixation」の諸問題について

https://www.ipa.go.jp/security/vuln/event/documents/20060228_3.pdf

1.5 クロスサイト・スクリプティング

ウェブアプリケーションの中には、検索のキーワードの表示画面や個人情報登録時の確認画面、掲示板、ウェブのログ統計画面等、利用者からの入力内容や HTTP ヘッダの情報を処理し、ウェブページとして出力するものがあります。ここで、ウェブページへの出力処理に問題がある場合、そのウェブページにスクリプト等を埋め込まれてしまいます。この問題を「クロスサイト・スクリプティングの脆弱性」と呼び、この問題を悪用した攻撃手法を、「クロスサイト・スクリプティング攻撃」と呼びます。クロスサイト・スクリプティング攻撃の影響は、ウェブサイト自体に対してではなく、そのウェブサイトのページを閲覧している利用者及びます。



■ 発生しうる脅威

クロスサイト・スクリプティング攻撃により、発生しうる脅威は次のとおりです。

- 本物サイト上に偽のページが表示される

- ・ 偽情報の流布による混乱
- ・ フィッシング詐欺による重要情報の漏えい 等

- ブラウザが保存している Cookie を取得される

- ・ Cookie にセッション ID が格納されている場合、さらに利用者へのなりすましにつながる¹⁵
- ・ Cookie に個人情報等が格納されている場合、その情報が漏えいする

¹⁵ 「1.4 セッション管理の不備」で解説した「発生しうる脅威」と同じ内容です。

- 任意の Cookie をブラウザに保存させられる

- ・ セッション ID が利用者に送り込まれ、「セッション ID の固定化¹⁶」攻撃に悪用される

■ 注意が必要なウェブサイトの特徴

運営主体やウェブサイトの性質を問わず、あらゆるサイトにおいて注意が必要な問題です。Cookie を利用してログインのセッション管理を行っているサイトや、フィッシング詐欺の攻撃ターゲットになりやすいページ(ログイン画面、個人情報の入力画面等)を持つサイトは、特に注意が必要です。

- この脆弱性が生じやすいページの機能例

- ・ 入力内容を確認させる表示画面(会員登録、アンケート等)
- ・ 誤入力時の再入力を要求する画面で、前の入力内容を表示するとき
- ・ 検索結果の表示
- ・ エラー表示
- ・ コメントの反映(ブログ、掲示板等) 等

■ 届出状況

クロスサイト・スクリプティングの脆弱性の届出件数は、他の脆弱性に比べて多くなっています。この脆弱性については、届出受付開始から 2014 年第 4 四半期までに、ウェブサイトの届出件数の約 5 割に相当する届出を受けています。また、ソフトウェア製品においても、この脆弱性に関して多数の届出を受けています。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・「iLogScanner」におけるクロスサイト・スクリプティングの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2014-000133>

- ・「Aflax」におけるクロスサイト・スクリプティングの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2014-000122>

- ・「Movable Type」におけるクロスサイト・スクリプティングの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2014-000104>

■ 対策について

クロスサイト・スクリプティングの脆弱性への対策は、ウェブアプリケーションの性質に合わせ、下記の 3 つに分類しています。

1) HTML テキストの入力を許可しない場合の対策

2) HTML テキストの入力を許可する場合の対策

3) 全てのウェブアプリケーションに共通の対策

1) に該当するウェブアプリケーションの例には、検索機能や個人情報の登録等、HTML タグ等を用いた入力を許可する必要がないものが挙げられます。多くのウェブアプリケーションがこちらに該当するはずですが。

¹⁶ 「セッション ID の固定化」については、p16 を参照してください。

2) に該当するウェブアプリケーションの例には、自由度の高い掲示板やブログ等が挙げられます。たとえば、利用者が入力文字の色やサイズを指定できる機能等を実装するために、HTML テキストの入力を許可する場合があるかもしれません。

3) は、1)、2) の両者のウェブアプリケーションに共通して必要な対策です。

1.5.1 HTML テキストの入力を許可しない場合の対策

■ 根本的解決

5-(i)

👉 **ウェブページに出力する全ての要素に対して、エスケープ処理を施す。**

ウェブページを構成する要素として、ウェブページの本文や HTML タグの属性値等に相当する全ての出力要素にエスケープ処理を行います。エスケープ処理には、ウェブページの表示に影響する特別な記号文字(「<」、「>」、「&」等)を、HTML エンティティ(「<」、「>」、「&」等)に置換する方法があります。また、HTML タグを出力する場合は、その属性値を必ず「"」(ダブルクォート)で括弧のようにします。そして、「"」で括弧された属性値に含まれる「"」を、HTML エンティティ「"」にエスケープします。

脆弱性防止の観点からエスケープ処理が必須となるのは、外部からウェブアプリケーションに渡される「入力値」の文字列や、データベースやファイルから読み込んだ文字列、その他、何らかの文字列を演算によって生成した文字列等です。しかし、必須であるか不必要であるかによらず、テキストとして出力するすべてに対してエスケープ処理を施すよう、一貫したコーディングをすることで、対策漏れ¹⁷を防止することができます。

なお、対象となる出力処理は HTTP レスポンスへの出力に限りません。JavaScript の document.write メソッドや innerHTML プロパティ等を使用して動的にウェブページの内容を変更する場合も、上記と同様の処理が必要です。

5-(ii)

👉 **URL を出力するときは、「http://」や「https://」で始まる URL のみを許可する。**

URL には、「http://」や「https://」から始まるものだけでなく、「javascript:」の形式で始まるものもあります。ウェブページに出力するリンク先や画像の URL が、外部からの入力に依存する形で動的に生成される場合、その URL にスクリプトが含まれていると、クロスサイト・スクリプティング攻撃が可能となる場合があります。たとえば、利用者から入力されたリンク先の URL を「」の形式でウェブページに出力するウェブアプリケーションは、リンク先の URL に「javascript:」等から始まる文字列を指定された場合に、スクリプトを埋め込まれてしまう可能性があります。リンク先の URL には「http://」や「https://」から始まる文字列のみを許可する、「ホワイトリスト方式」で実装してください。

¹⁷ 3.5.2 対策漏れを参照。

5-(iii)

☞ **<script>...</script> 要素の内容を動的に生成しない。**

ウェブページに出力する<script>...</script>要素の内容が、外部からの入力に依存する形で動的に生成される場合、任意のスクリプトが埋め込まれてしまう可能性があります。危険なスクリプトだけを排除する方法も考えられますが、危険なスクリプトであることを確実に判断することは難しいため、<script>...</script>要素の内容を動的に生成する仕様は、避けることをお勧めします。

5-(iv)

☞ **スタイルシートを任意のサイトから取り込めるようにしない。**

スタイルシートには、expression() 等を利用してスクリプトを記述することができます。このため任意のサイトに置かれたスタイルシートを取り込めるような設計をすると、生成するウェブページにスクリプトが埋め込まれてしまう可能性があります。取り込んだスタイルシートの内容をチェックし、危険なスクリプトを排除する方法も考えられますが、確実に排除することは難しいため、スタイルシートを外部から指定可能な仕様は、避けることが望まれます。

■ 保険的対策

5-(v)

☞ **入力値の内容チェックを行う。**

入力値すべてについて、ウェブアプリケーションの仕様に沿うものかどうかを確認する処理を実装し、仕様に合わない値を入力された場合は処理を先に進めず、再入力を求めるようにする対策方法です。ただし、この対策が有効となるのは限定的です。例えば、アプリケーションの要求する仕様が幅広い文字種の入力を許すものである場合には対策にならないため、この方法に頼ることはお勧めできません。

対策になるとすれば、アプリケーションの要求する仕様が英数字のみの入力を許すものである場合などであり、この仕様の入力についての内容チェックはクロスサイト・スクリプティング攻撃を防止できる可能性が高いですが、この場合も、入力値の確認処理を通過した後の文字列の演算結果がスクリプト文字列を形成してしまうプログラムとなっている可能性を想定すれば、やはり完全な対策ではありません。

1.5.2 HTML テキストの入力を許可する場合の対策

■ 根本的解決

5-(vi)

☞ **入力された HTML テキストから構文解析木を作成し、スクリプトを含まない必要な要素のみを抽出する**

入力された HTML テキストに対して構文解析を行い、「ホワイトリスト方式」で許可する要素のみを抽出します。ただし、これには複雑なコーディングが要求され、処理に負荷がかかるといった影響もある

ため、実装には十分な検討が必要です。

■ 保険的対策

5-(vii)

👉 **入力された HTML テキストから、スクリプトに該当する文字列を排除する。**

入力された HTML テキストに含まれる、スクリプトに該当する文字列を抽出し、排除してください。抽出した文字列の排除方法には、無害な文字列へ置換することをお勧めします。たとえば、「<script>」や「javascript:」を無害な文字列へ置換する場合、「<xscript>」「xjavascript:」のように、その文字列に適当な文字を付加します。他の排除方法として、文字列の削除が挙げられますが、削除した結果が危険な文字列を形成してしまう可能性¹⁸があるため、お勧めできません。

なお、この対策は、危険な文字列を完全に抽出することが難しいという問題があります。ウェブブラウザによっては、「java	script:」や「java(改行コード)script:」等の文字列を「javascript:」と解釈してしまうため、単純なパターンマッチングでは危険な文字列を抽出することができません。そのため、このような「ブラックリスト方式」による対策のみに頼ることはお勧めできません。

1.5.3 全てのウェブアプリケーションに共通の対策

■ 根本的解決

5-(viii)

👉 **HTTP レスポンスヘッダの Content-Type フィールドに文字コード(charset)を指定する。**

HTTP のレスポンスヘッダの Content-Type フィールドには、「Content-Type: text/html; charset=UTF-8」のように、文字コード(charset)を指定できます。この指定を省略した場合、ブラウザは、文字コードを独自の方法で推定して、推定した文字コードにしたがって画面表示を処理します。たとえば、一部のブラウザにおいては、HTML テキストの冒頭部分等に特定の文字列が含まれていると、必ず特定の文字コードとして処理されるという挙動が知られています。

Content-Type フィールドで文字コードの指定を省略した場合、攻撃者が、この挙動を悪用して、故意に特定の文字コードをブラウザに選択させるような文字列を埋め込んだ上、その文字コードで解釈した場合にスクリプトのタグとなるような文字列を埋め込む可能性があります。

たとえば、具体的な例として、HTML テキストに、
「+ADw-script+AD4-alert(+ACI-test+ACI-)+ADsAPA-/script+AD4-」という文字列が埋め込まれた場合が考えられます。この場合、一部のブラウザはこれを「UTF-7」の文字コードでエンコードされた文字列として識別します。これが UTF-7 として画面に表示されると
「<script>alert('test');</script>」として扱われるため、スクリプトが実行されてしまいます。

ウェブアプリケーションが、前記 5-(i) の「エスケープ処理」を施して正しくクロスサイト・スクリプティングの脆弱性への対策をしている場合であっても、本来対象とする文字が UTF-8 や EUC-JP、

¹⁸ 3.5.3 のよくある失敗例 2 を参照。

Shift_JIS 等の文字コードで扱われてしまうと、「+ADw-」等の文字列が「エスケープ処理」されることはありません。

この問題への対策案として、「エスケープ処理」の際に UTF-7 での処理も施すという方法が考えられますが、UTF-7 のみを想定すれば万全とは言い切れません。またこの方法では、UTF-7 を前提に「エスケープ処理」した結果、正当な文字列(たとえば「+ADw-」という文字列)が別の文字列になるという、本来の機能に支障をきたすという不具合が生じます。

したがって、この問題の解決策としては、Content-Type の出力時に charset を省略することなく、必ず指定することが有効です。ウェブアプリケーションが HTML 出力時に想定している文字コードを、Content-Type の charset に必ず指定してください¹⁹。

■ 保険的対策

5-(ix)

👉 **Cookie 情報の漏えい対策として、発行する Cookie に HttpOnly 属性を加え、TRACE メソッドを無効化する。**

「HttpOnly」は、Cookie に設定できる属性のひとつで、これが設定された Cookie は、HTML テキスト内のスクリプトからのアクセスが禁止されます。これにより、ウェブサイトクロスサイト・スクリプティングの脆弱性が存在する場合であっても、その脆弱性によって Cookie を盗まれるという事態を防止できます。

具体的には、Cookie を発行する際に、「Set-Cookie:(中略)HttpOnly」として設定します。なお、この対策を採用する場合には、いくつかの注意が必要²⁰です。

HttpOnly 属性は、ブラウザによって対応状況に差がある²¹ため、全てのウェブサイト閲覧者に有効な対策ではありません。

本対策は、クロスサイト・スクリプティングの脆弱性のすべての脅威をなくすものではなく、Cookie 漏えい以外の脅威は依然として残るものであること、また、利用者のブラウザによっては、この対策が有効に働かない場合があることを理解した上で、対策の実施を検討してください。

¹⁹ W3C 勧告 HTML 4.0.1 では、ブラウザに対し、文字コードを決定する場合には次の優先順位を守らねばならない、としています(<http://www.w3.org/TR/html401/charset.html#h-5.2.2>)。

1. HTTP ヘッダの Content-Type フィールドの charset パラメータ
2. META 要素で、http-equiv 属性値が Content-Type かつ value 属性の値に charset 情報があるもの
3. 外部リソースを指している要素に設定されている charset 属性値

したがって、文字コードの指定箇所は、1.の「HTTP ヘッダの Content-Type フィールドの charset パラメータ」であることが望ましいと考えられます。

²⁰ Windows XP 以前ではウェブサーバにおいて「TRACE メソッド」を無効とする必要がありました。「TRACE メソッド」が有効である場合、サイトにクロスサイト・スクリプティングの脆弱性があると、「Cross-Site Tracing」と呼ばれる攻撃手法によって、ブラウザから送信される HTTP リクエストヘッダの全体が取得されてしまいます。HTTP リクエストヘッダには Cookie 情報も含まれるため、HttpOnly 属性を加えていても Cookie は取得されてしまいます。

²¹ HttpOnly に対する各ブラウザの対応状況は、下記のページを参照してください。

Browserscope: <http://www.browserscope.org/?category=security>

5-(x)

👉 **クロスサイト・スクリプティングの潜在的な脆弱性対策として有効なブラウザの機能を有効にするレスポンスヘッダを返す。**

ブラウザには、クロスサイト・スクリプティング攻撃のブロックを試みる機能を備えたものがあります。しかし、ユーザの設定によっては無効になってしまっている場合があるため、サーバ側から明示的に有効にするレスポンスヘッダを返すことで、ウェブアプリケーションにクロスサイト・スクリプティング脆弱性があった場合にも悪用を避けることができます。ただし、下記に示すレスポンスヘッダは、いずれもブラウザによって対応状況に差がある²²ため、全てのウェブサイト閲覧者に有効な対策ではありません。

【X-XSS-Protection】

「X-XSS-Protection²³」は、ブラウザの「XSS フィルタ」の設定を有効にするパラメータです。ブラウザで明示的に無効になっている場合でも、このパラメータを受信することで有効になります。HTTP レスポンスヘッダに「X-XSS-Protection: 1; mode=block」のように設定することで、クロスサイト・スクリプティング攻撃のブロックを試みる機能が有効になります。

【Content Security Policy】

「Content Security Policy (CSP)²⁴」は、ブラウザで起こりうる問題を緩和するセキュリティの追加レイヤーです。その機能の一つに、反射型クロスサイト・スクリプティング攻撃を防止する「reflected-xss」があります。HTTP レスポンスヘッダに「Content-Security-Policy: reflected-xss block」のように設定することで、クロスサイト・スクリプティング攻撃のブロックを試みる機能が有効になります。

以上の対策により、クロスサイト・スクリプティング攻撃に対する安全性の向上が期待できます。クロスサイト・スクリプティングの脆弱性に関する情報については、次の資料も参考にしてください。

■ CWE

CWE-79: クロスサイト・スクリプティング (XSS)

<https://jvndb.jvn.jp/ja/cwe/CWE-79.html>

CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

<https://cwe.mitre.org/data/definitions/79.html>

²² X-XSS-Protection および Content Security Policy に対する各ブラウザの対応状況は、下記のページを参考にしてください。

Browserscope: <http://www.browserscope.org/?category=security>

²³ IEInternals: Controlling the XSS Filter

<http://blogs.msdn.com/b/ieinternals/archive/2011/01/31/controlling-the-internet-explorer-xss-filter-with-the-x-xss-protection-http-header.aspx>

²⁴ Content Security Policy Level 2

<http://w3c.org/TR/CSP2/>

■ 参考 URL

IPA: 知っていますか?脆弱性(ぜいじゃくせい)「2. クロスサイト・スクリプティング」

https://www.ipa.go.jp/security/vuln/vuln_contents/xss.html

IPA: テクニカルウォッチ『DOM Based XSS』に関するレポート

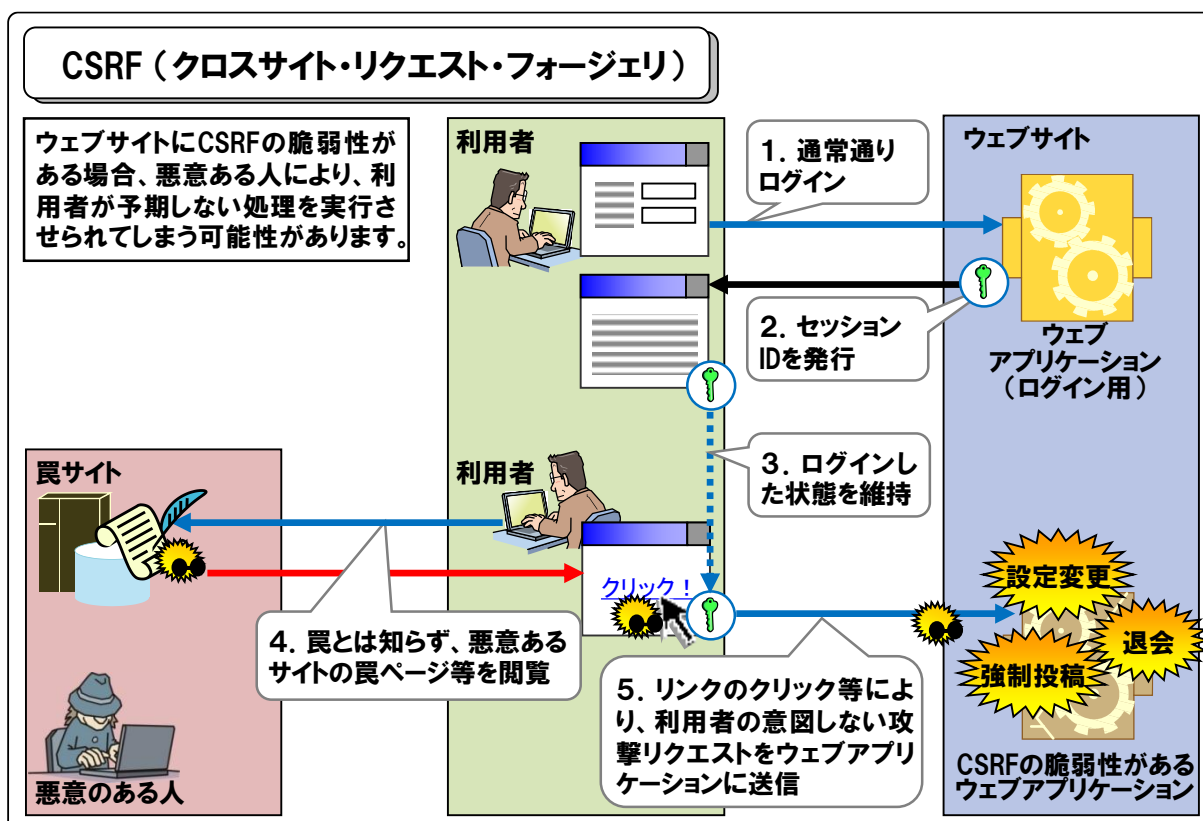
<https://www.ipa.go.jp/about/technicalwatch/20130129.html>

JPCERT/CC: HTML5 を利用した Web アプリケーションのセキュリティ問題に関する調査報告書

<https://www.jpcert.or.jp/research/html5.html>

1.6 CSRF(クロスサイト・リクエスト・フォージェリ)

ウェブサイトの中には、サービスの提供に際しログイン機能を設けているものがあります。ここで、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、外部サイトを經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。このような問題を「CSRF (Cross-Site Request Forgeries / クロスサイト・リクエスト・フォージェリ)の脆弱性」と呼び、これを悪用した攻撃を、「CSRF 攻撃」と呼びます。



■ 発生しうる脅威

CSRF 攻撃により、発生しうる脅威²⁵ は次のとおりです。

- ログイン後の利用者のみが利用可能なサービスの悪用

不正な送金、利用者が意図しない商品購入、利用者が意図しない退会処理 等

- ログイン後の利用者のみが編集可能な情報の改ざん、新規登録

各種設定の不正な変更(管理者画面、パスワード等)、掲示板への不適切な書き込み 等

²⁵ 前述「1.4 セッション管理の不備」における脅威と比較してみると、攻撃者は、「ログインした利用者のみが閲覧可能な情報」を閲覧することができない、という違いがあると言えます。ただし、「パスワード変更」のように、次の攻撃(なりすまし)に繋がる攻撃が成功した場合には、情報漏えいの脅威も発生する可能性があります。

■ 注意が必要なウェブサイトの特徴

次の技術を利用してセッション管理を実装しているウェブサイトが、CSRF 攻撃による影響を受ける可能性があります。

- Cookie を用いたセッション管理
- Basic 認証
- SSL クライアント認証

また、上記を実装するウェブサイトのうち、ログイン後に決済処理等の重要な処理を行うサイトは、攻撃による被害が大きくなるため、特に注意が必要です。

- 金銭処理が発生するサイト
 - ネットバンキング、ネット証券、ショッピング、オークション 等
- その他、ログイン機能を持つサイト
 - 管理画面、会員専用サイト、日記サイト 等

■ 届出状況

CSRF の脆弱性に関する届出が、ウェブサイトの届出全体に占める割合は、1パーセント未満と多くはありません。しかしながらこれらの脆弱性については、ソフトウェア製品の届出を含め、2006 年頃から継続的に届出を受けています。届出の報告内容としては、ネットワーク対応ハードディスク等、組み込み製品のウェブ管理画面に同脆弱性が存在する例等があります。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・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
<https://jvndb.jvn.jp/jvndb/JVNDDB-2015-000012>
- ・「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
<https://jvndb.jvn.jp/jvndb/JVNDDB-2014-000064>
- ・「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
<https://jvndb.jvn.jp/jvndb/JVNDDB-2013-000097>

■ 根本的解決

6-(i)-a

- ☞ 処理を実行するページを POST メソッドでアクセスするようにし、その「hidden パラメータ」に秘密情報が挿入されるよう、前のページを自動生成して、実行ページではその値が正しい場合のみ処理を実行する。

ここでは具体的な例として、「入力画面 → 確認画面 → 登録処理」のようなページ遷移を取り上げて説明します。まず、利用者の入力内容を確認画面として出力する際、合わせて秘密情報を「hidden パラメータ」に出力するようにします。この秘密情報は、セッション管理に使用しているセッション ID を用いる方法の他、セッション ID とは別のもうひとつの ID (第 2 セッション ID) をログイン時に生成

して用いる方法等が考えられます。生成する ID は暗号論的擬似乱数生成器を用いて、第三者に予測困難なように生成する必要があります。次に確認画面から登録処理のリクエストを受けた際は、リクエスト内容に含まれる「hidden パラメータ」の値と、秘密情報とを比較し、一致しない場合は登録処理を行わないようにします²⁶。このような実装であれば、攻撃者が「hidden パラメータ」に出力された秘密情報を入力できなければ、攻撃は成立しません。

なお、このリクエストは、POST メソッドで行うようにします²⁷。これは、GET メソッドで行った場合、外部サイトに送信される Referer に秘密情報が含まれてしまうためです。

6-(i)-b

👉 **処理を実行する直前のページで再度パスワードの入力を求め、実行ページでは、再度入力されたパスワードが正しい場合のみ処理を実行する。**

処理の実行前にパスワード認証を行うことにより、CSRF の脆弱性を解消できます²⁸。ただし、この方法は画面設計の仕様変更を要する対策であるため、画面設計の仕様変更をせず、実装の変更だけで対策をする必要がある場合には、6-(i)-a または 6-(i)-c の対策を検討してください。

この対策方法は、上記 6-(i)-a と比べて実装が簡単となる場合があります。たとえば、セッション管理の仕組みを使用しないで Basic 認証を用いている場合、6-(i)-a の対策をするには新たに秘密情報を作る必要があります。このとき、暗号論的擬似乱数生成器を簡単には用意できないならば、この対策の方が採用しやすいと言えます。

6-(i)-c

👉 **Referer が正しいリンク元かを確認し、正しい場合のみ処理を実行する。**

Referer を確認することにより、本来の画面遷移を経ているかどうかを判断できます。Referer が確認できない場合は、処理を実行しないようにします²⁹。また Referer が空の場合も、処理を実行しないようにします。これは、Referer を空にしてページを遷移する方法が存在し、攻撃者がその方法を利用して CSRF 攻撃を行う可能性があるためです。

ただし、ウェブサイトによっては、攻撃者がそのウェブサイト上に罫を設置することができる場合があります。このようなサイトでは、この対策法が有効に機能しない場合があります。また、この対策法を採用すると、ブラウザやパーソナルファイアウォール等の設定で Referer を送信しないようにしている利用者が、そのサイトを利用できなくなる不都合が生じる可能性があります。本対策の採用には、これらの点にも注意してください。

²⁶ 3.6 の修正例 1 を参照。

²⁷ HTTP/1.1 の仕様を定義している RFC2616 には、「機密性の求められるデータの送信には GET メソッドを使わず、POST メソッドを使うべきである」という内容の記述があります (15.1.3 Encoding Sensitive Information in URI's)。

RFC2616: 「Hypertext Transfer Protocol -- HTTP/1.1」 <http://www.ietf.org/rfc/rfc2616.txt>

²⁸ 3.6 の修正例 2 を参照。

²⁹ 3.6 の修正例 3 を参照。

■ 保険的対策

6-(ii)

👉 **重要な操作を行った際に、その旨を登録済みのメールアドレスに自動送信する。**

メールの通知は事後処理であるため、CSRF 攻撃自体は防ぐことはできません。しかしながら、実際に攻撃があった場合に、利用者が異変に気付くきっかけを作ることができます。なお、メール本文には、プライバシーに関わる重要な情報を入れない等の注意が必要です。

以上の対策により、CSRF 攻撃に対する安全性の向上が期待できます。CSRF の脆弱性に関する情報については、次の資料も参考にしてください。

■ CWE

CWE-352: クロスサイト・リクエスト・フォージェリ(CSRF)

<https://jvndb.jvn.jp/ja/cwe/CWE-352.html>

CWE-352: Cross-Site Request Forgery (CSRF)

<https://cwe.mitre.org/data/definitions/352.html>

■ 参考 URL

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「3. CSRF (クロスサイト・リクエスト・フォージェリ)」

https://www.ipa.go.jp/security/vuln/vuln_contents/csrf.html

産業技術総合研究所 高木浩光: 「CSRF」と「Session Fixation」の諸問題について

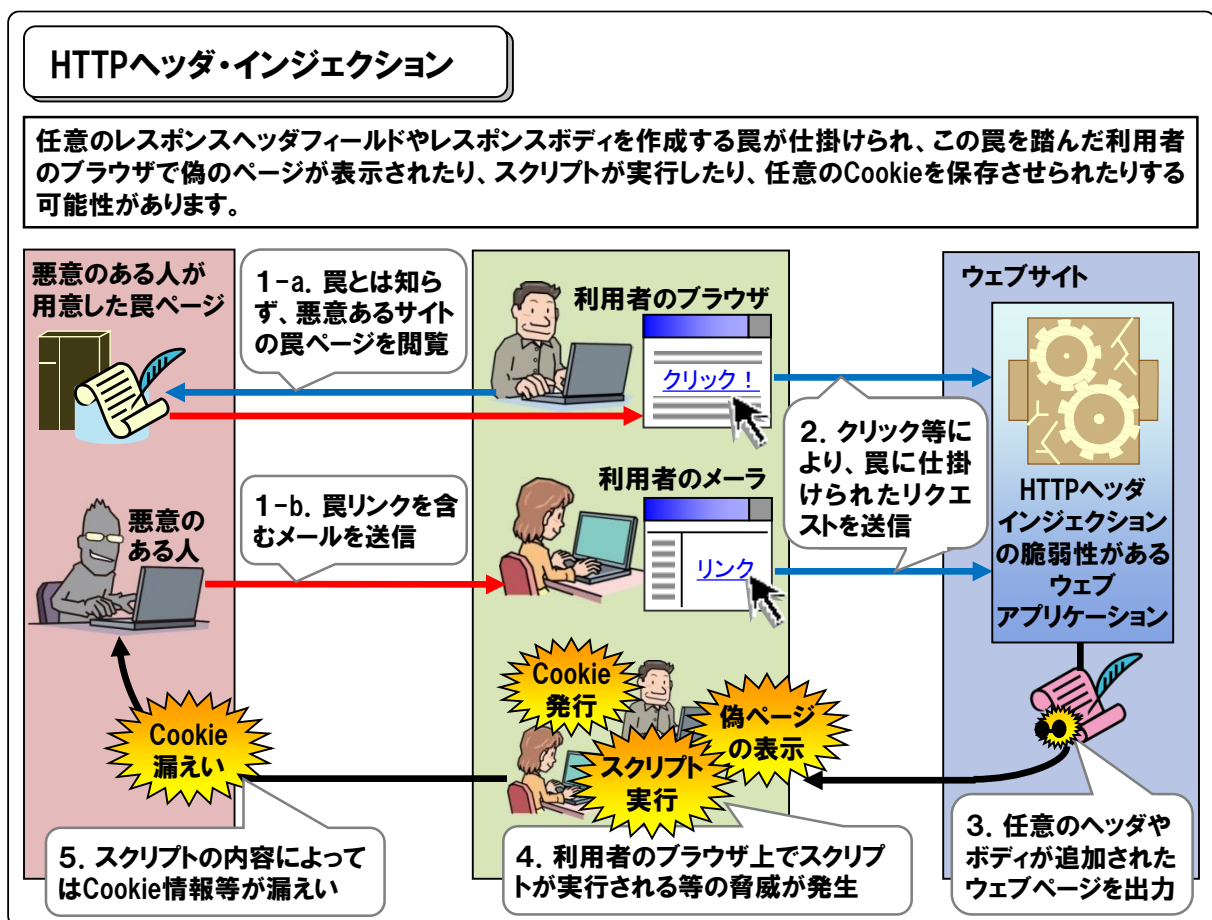
https://www.ipa.go.jp/security/vuln/event/documents/20060228_3.pdf

JPCERT/CC: HTML5 を利用した Web アプリケーションのセキュリティ問題に関する調査報告書

<https://www.jpCERT.or.jp/research/html5.html>

1.7 HTTP ヘッダ・インジェクション

ウェブアプリケーションの中には、リクエストに対して出力する HTTP レスポンスヘッダのフィールド値を、外部から渡されるパラメータの値等を利用して動的に生成するものがあります。たとえば、HTTP リダイレクションの実装として、パラメータから取得したジャンプ先の URL 情報を、Location ヘッダのフィールド値に使用する場合や、掲示板等において入力された名前等を Set-Cookie ヘッダのフィールド値に使用する場合等が挙げられます。このようなウェブアプリケーションで、HTTP レスポンスヘッダの出力処理に問題がある場合、攻撃者は、レスポンス内容に任意のヘッダフィールドを追加したり、任意のボディを作成したり、複数のレスポンスを作り出すような攻撃を仕掛ける場合があります。このような問題を「HTTP ヘッダ・インジェクションの脆弱性」と呼び、この問題を悪用した攻撃手法は「HTTP ヘッダ・インジェクション攻撃」と呼びます。特に、複数のレスポンスを作り出す攻撃は、「HTTP レスポンス分割(HTTP Response Splitting) 攻撃」と呼びます。



■ 発生しうる脅威

本脆弱性を突いた攻撃により、発生しうる脅威は次のとおりです。

- クロスサイト・スクリプティング攻撃により発生しうる脅威と同じ脅威

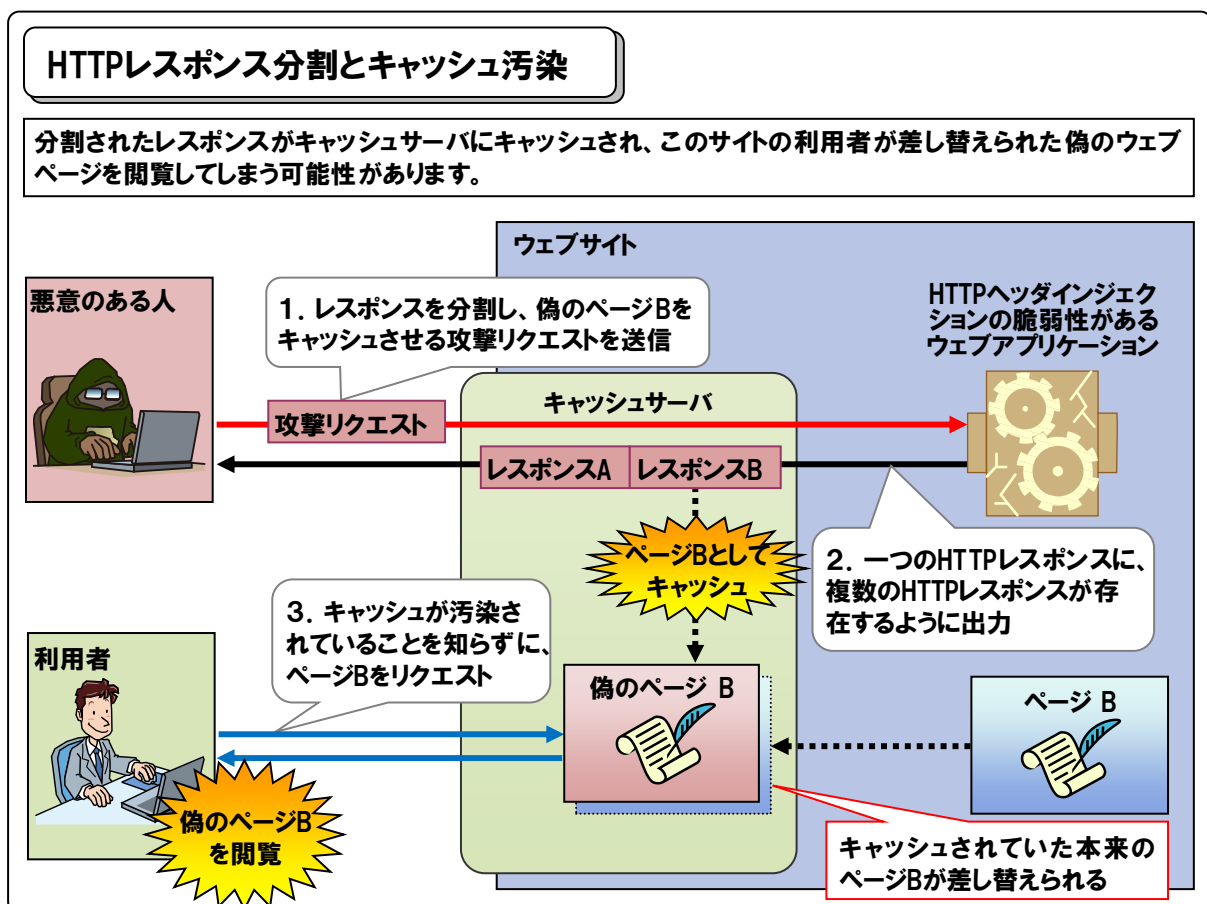
任意のレスポンスボディを注入された場合、利用者のブラウザ上で偽の情報を表示させられたり、任意のスクリプトを埋め込まれたりする可能性があります。これは、前述「1.5 クロスサイト・スクリプティング」で解説した「発生しうる脅威」と同じ脅威です。

- 任意の Cookie 発行

Set-Cookie ヘッダを注入された場合、任意の Cookie が発行され、利用者のブラウザに保存される可能性があります。

- キャッシュサーバのキャッシュ汚染

複数のレスポンスに分割し、任意のレスポンスボディをリバースプロキシ等にキャッシュさせることにより、キャッシュ汚染(ウェブページの差し替え)を引き起こし、ウェブページの改ざんと同じ脅威が生じます。この攻撃を受けたウェブサイトにはアクセスする利用者は、この差し替えられた偽のウェブページを参照し続けることとなります。クロスサイト・スクリプティング攻撃のように、攻撃を受けた直後の本人のみが影響を受ける場合に比べ、キャッシュ汚染による脅威は、影響を受ける対象が広く、また永続的であることが特徴です。



■ 注意が必要なウェブサイトの特徴

運営主体やウェブサイトの性質を問わず、HTTP レスポンスヘッダのフィールド値 (Location ヘッダ、Set-Cookie ヘッダ等) を、外部から渡されるパラメータの値から動的に生成する実装のウェブアプリケーションに注意が必要な問題です。Cookie を利用してログインのセッション管理を行っているサイトや、サイト内にリバースプロキシとしてキャッシュサーバを構築しているサイトは、特に注意が必要です。

■ 届出状況

HTTP ヘッダ・インジェクションの脆弱性の届出がウェブサイトの届出全体に占める割合は数パーセントと多くはありません。しかしながらこれらの脆弱性については受付開始当初から継続的に届出を受けています。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・「Pebble」における HTTP ヘッダ・インジェクションの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDB-2012-000099>
- ・「Cogent DataHub」における HTTP ヘッダ・インジェクションの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDB-2012-000002>
- ・「Active! mail 6」における HTTP ヘッダ・インジェクションの脆弱性
<https://jvndb.jvn.jp/jvndb/JVNDB-2010-000050>

■ 根本的解決

7-(i)-a

👉 **ヘッダの出力を直接行わず、ウェブアプリケーションの実行環境や言語に用意されているヘッダ出力用 API を使用する。**

ウェブアプリケーションの実行環境によっては、Content-Type フィールドをはじめとする HTTP レスポンスヘッダを、プログラムで直接出力するものがあります。このような場合に、フィールド値に式の値をそのまま出力すると、外部から与えられた改行コードが余分な改行として差し込まれることとなります。HTTP ヘッダは改行によって区切られる構造となっているため、これを許すと、任意のヘッダフィールドや任意のボディを注入されたり、レスポンスを分割されたりする原因となります。ヘッダの構造は継続行が許される等単純なものではありませんので、実行環境に用意されたヘッダ出力用の API を使用することをお勧めします。

ただし、実行環境によっては、ヘッダ出力 API が改行コードを適切に処理しない脆弱性が指摘されているものもあります。その場合には修正パッチを適用するか、適用できない場合には、次の 7-(i)-b または 7-(ii) の対策をとります。

7-(i)-b

👉 **改行コードを適切に処理するヘッダ出力用 API を利用できない場合は、改行を許可しないよう、開発者自身で適切な処理を実装する。**

例えば、改行の後に空白を入れることで継続行として処理する方法や、改行コード以降の文字を削

除する方法³⁰、改行が含まれていたらウェブページ生成の処理を中止する方法等が考えられます。

■ 保険的対策

7-(ii)

☞ 外部からの入力の全てについて、改行コードを削除する。

外部からの入力の全てについて、改行コードを削除します。あるいは、改行コードだけでなく、制御コード全てを削除してもよいかもしれません。ただし、ウェブアプリケーションが、TEXTAREA の入力データ等、改行コードを含みうる文字列を受け付ける必要がある場合には、この対策のように一律に全ての入力に対して処理を行うと、対策を実施したウェブアプリケーションが正しく動作しなくなるため、注意が必要です。

以上の対策により、HTTP ヘッダ・インジェクション攻撃に対する安全性の向上が期待できます。HTTP ヘッダ・インジェクションの脆弱性に関する情報については、次の資料も参考にしてください。

■ CWE

CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
<https://cwe.mitre.org/data/definitions/113.html>

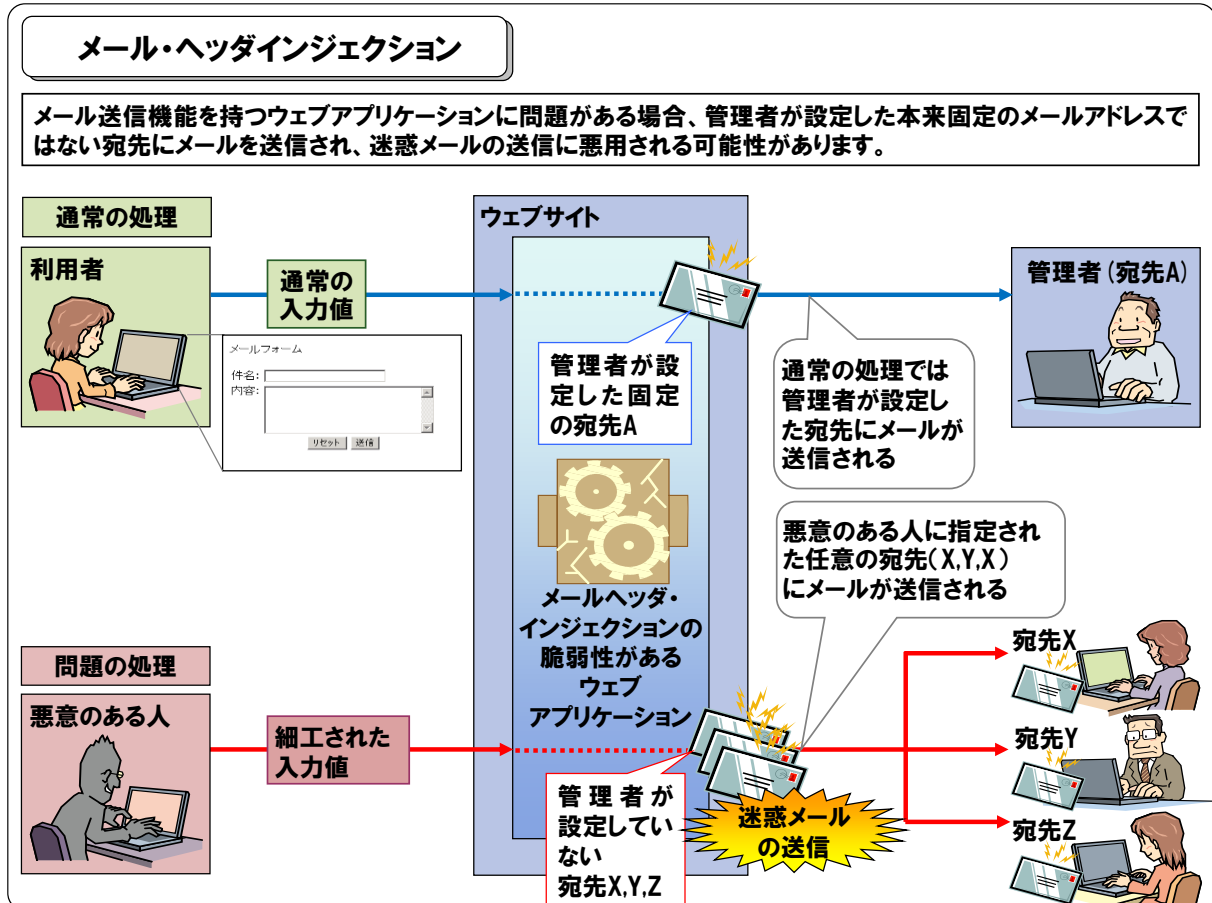
■ 参考 URL

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「7. HTTP ヘッダ・インジェクション」
https://www.ipa.go.jp/security/vuln/vuln_contents/hhi.html

³⁰ 3.7 の修正例を参照。

1.8 メールヘッダ・インジェクション

ウェブアプリケーションの中には、利用者が入力した商品申し込みやアンケート等の内容を、特定のメールアドレスに送信する機能を持つものがあります。一般に、このメールアドレスは固定で、ウェブアプリケーションの管理者以外の人に変更できませんが、実装によっては、外部の利用者がこのメールアドレスを自由に指定できてしまう場合があります。このような問題を引き起こす脆弱性を「メールヘッダ・インジェクション」と呼び、それを悪用した攻撃を、「メールヘッダ・インジェクション攻撃」と呼びます。



■ 発生しうる脅威

メールヘッダ・インジェクション攻撃が行われた場合、発生しうる脅威は次のとおりです。

- メールの中継

迷惑メールの送信に悪用される

■ 注意が必要なウェブサイトの特徴

利用者が入力した内容を管理者宛にメールで送信する機能を実装しているウェブサイトが、「メールの第三者中継」による影響を受けます。該当する機能には、「問い合わせページ」や「アンケート」等があります。

■ 届出状況

メールの第三者中継が可能な脆弱性の届出は、ウェブサイトの届出全体に占める割合は 1 パーセン

ト未滿と多くはありません。受付開始当初から断続的に届出を受けています。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・「サイボウズ ガルーン」におけるメールヘッダ・インジェクションの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDDB-2013-000116>

- ・CGI RESCUE 製「フォームメール」におけるメールの不正送信が可能な脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDDB-2009-000023>

- ・「MailDwarf」においてメールの不正送信が可能な脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDDB-2007-000229>

■ 根本的解決

8-(i)-a

☞ **メールヘッダを固定値にして、外部からの入力はすべてメール本文に出力する。**

「To」、「Cc」、「Bcc」、「Subject」等のメールヘッダの内容が外部からの入力に依存する場合や、メール送信プログラムへの出力処理に問題がある場合、外部からの入力をそのまま出力すると、外部から与えられた改行コードが余分な改行として差し込まれることとなります。これを許すと、任意のメールヘッダの挿入や、メール本文の改変、任意の宛先へのメール送信に悪用される原因となります。外部からの入力をメールヘッダに出力しない実装³¹をお勧めします。

8-(i)-b

☞ **メールヘッダを固定値にできない場合、ウェブアプリケーションの実行環境や言語に用意されているメール送信用 API を使用する。**

メールヘッダを固定値にできない場合の例としては、メールの件名を変更したい場合等があります。外部からの入力をメールヘッダに出力する場合、ウェブアプリケーションの実行環境や言語に用意されているメール送信用 API を使用することをお勧めします。ただし、API によっては改行コードの取り扱いが不適切なもの、複数のメールヘッダが挿入できる仕様のもがあります。その場合、脆弱性が修正されたバージョンを使用するか、改行を許可しないよう、開発者自身で適切な処理を実装します。改行を許可しない適切な処理には、改行コードの後に空白か水平タブを入れることで継続行として処理する方法や、改行コード以降の文字を削除する方法、改行が含まれていたらウェブページの生成の処理を中止する方法等が考えられます。

8-(ii)

☞ **HTML で宛先を指定しない。**

これは、いわば「論外」の実装ですが、hidden パラメータ等に宛先をそのまま指定するという事例の届出がありましたので、避けるべき実装として紹介します。

ウェブアプリケーションに渡されるパラメータに宛先を直接指定する実装は、パラメータ値の改変により、メールシステムの第三者中継につながる可能性があります。

³¹ 3.8 の修正例 1 を参照。

■ 保険的対策

8-(iii)

☞ 外部からの入力の全てについて、改行コードを削除する。

外部からの入力の全てについて、改行コードを削除します³²。あるいは改行コードだけではなく、制御コード全てを削除してもよいかもしれません。ただし、ウェブアプリケーションが、メール本文に出力するデータ等、改行コードを含みうる文字列にも、全ての入力に対して処理を行うと、そのウェブアプリケーションが正しく動作しなくなるため、注意が必要です。

以上の対策により、メールヘッダ・インジェクション攻撃に対する安全性の向上が期待できます。メールヘッダ・インジェクションに関する情報については、次の資料も参考にしてください。

■ CWE

CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')

<https://cwe.mitre.org/data/definitions/93.html>

■ 参考 URL

IPA: 知っていますか？脆弱性（ぜいじゃくせい）「10. メール不正中継」

https://www.ipa.go.jp/security/vuln/vuln_contents/mail.html

³² 3.8 の修正例 2 を参照。

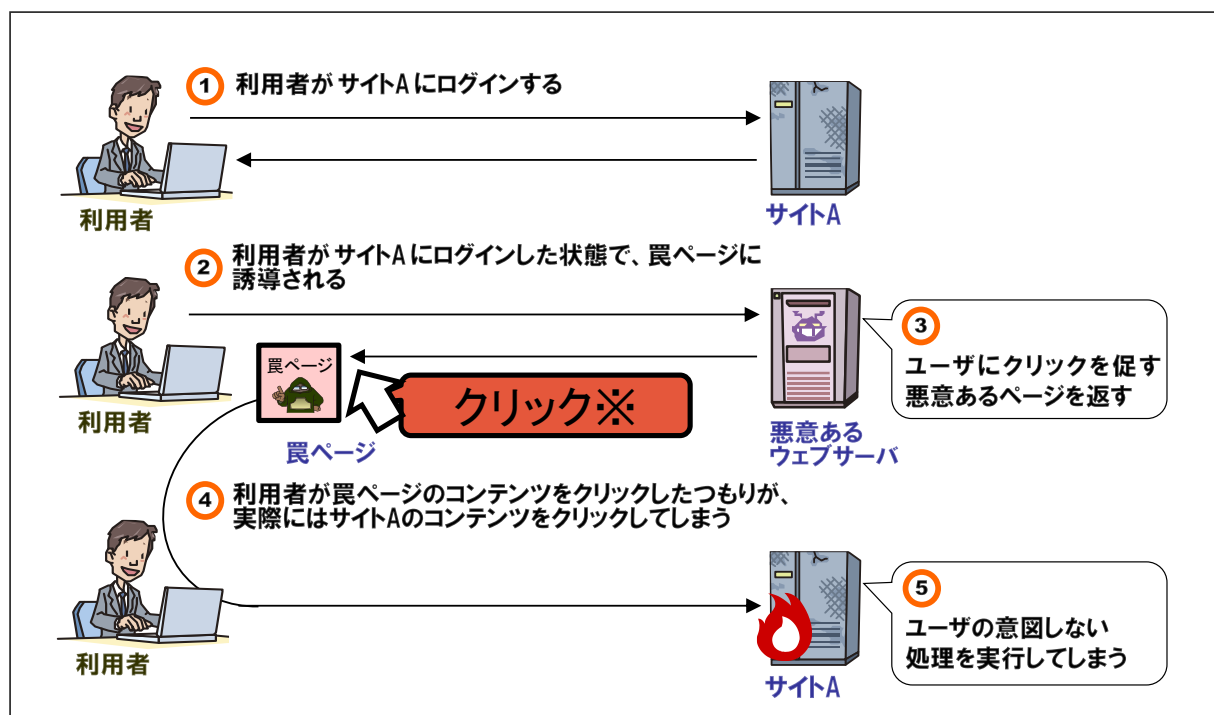
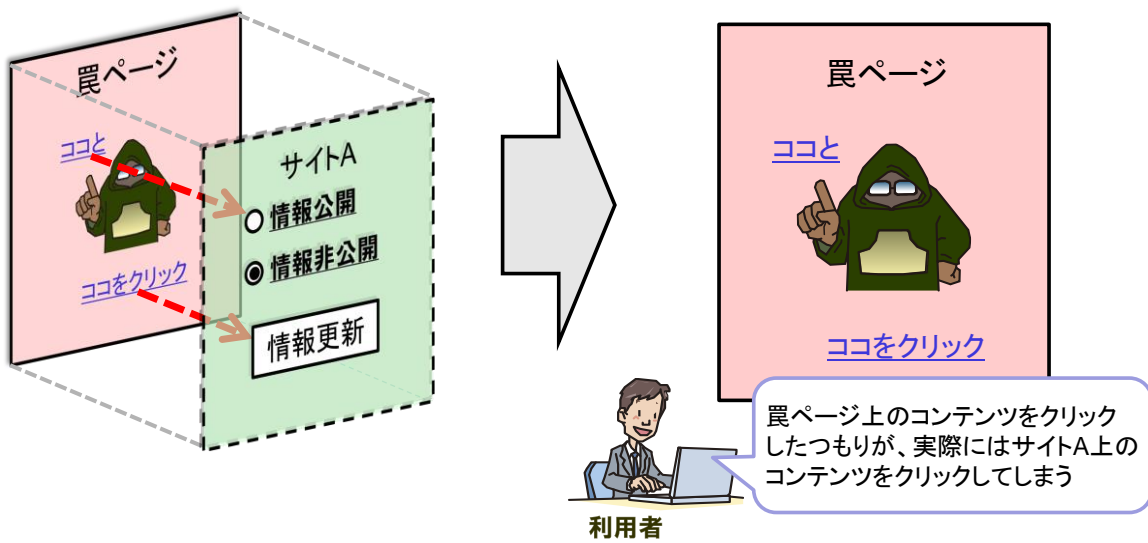
1.9 クリックジャッキング

ウェブサイトの中には、ログイン機能を設け、ログインしている利用者のみが使用可能な機能を提供しているものがあります。該当する機能がマウス操作のみで使用可能な場合、細工された外部サイトを閲覧し操作することにより、利用者が誤操作し、意図しない機能を実行させられる可能性があります。このような問題を「クリックジャッキングの脆弱性」と呼び、問題を悪用した攻撃を、「クリックジャッキング攻撃」と呼びます。

※罨ページの仕組み

① 罨ページの上に、サイトAをiframeで重ね合わせ、見た目を透明にする

② ブラウザから閲覧した際には、罨ページのみ表示されているように見える



■ 発生しうる脅威

クリックジャッキング攻撃により、発生しうる脅威は次のとおりです。マウス操作のみで実行可能な処理に限定される点以外は、CSRF 攻撃による脅威と同様です。

- ログイン後の利用者のみが利用可能なサービスの悪用

利用者が意図しない情報発信、利用者が意図しない退会処理 等

- ログイン後の利用者のみが編集可能な設定の変更

利用者情報の公開範囲の意図しない変更 等

■ 注意が必要なウェブサイトの特徴

ログイン後の利用者のみが利用可能な機能(サービスや設定)を、マウス操作のみで実行可能なウェブサイトが、クリックジャッキング攻撃による影響を受ける可能性があります。マウス操作のみで実行可能な処理が、利用者に紐づいた情報の公開範囲の変更処理等の場合は、攻撃による被害が大きくなるため、特に注意が必要です。

また、対策を実施した場合、後述する副作用が発生します。そのため、ウェブサイトの情報セキュリティポリシーや副作用等を加味して、クリックジャッキングの脆弱性対策の実施有無を検討してください。

■ 届出状況

クリックジャッキングの脆弱性に関するウェブサイトの届出は、2011 年に初めて受け付けました。ウェブサイトの届出全体に占める割合は、1 パーセント未満と多くはありません。しかしながらこれらの脆弱性については 2011 年から継続して届出を受けています。なお、届出受付開始から 2014 年第 4 四半期までに、ソフトウェア製品の届出は受け付けていません。

■ 根本的解決

9-(i)-a

☞ **HTTP レスポンスヘッダに、X-Frame-Options ヘッダフィールドを出力し、他ドメインのサイトからの frame 要素や iframe 要素による読み込みを制限する。**

X-Frame-Options は、ウェブアプリケーションをクリックジャッキング攻撃から防御するためのヘッダです³³。HTTP のレスポンスヘッダに「X-Frame-Options: DENY」のように出力することで、X-Frame-Options に対応したブラウザにおいて、frame 要素や iframe 要素によるページ読み込みの制限ができます。なお、Internet Explorer 7 は、X-Frame-Options ヘッダに対応していないため、本対策を実施したとしても、当該ブラウザにおいてはクリックジャッキング攻撃を防げません。

X-Frame-Options で指定する設定値により、制限の範囲が変わります。設定値の挙動は下記表の通りです。なお、ALLOW-FROM はブラウザによって適切に動作しない場合があります。開発しているウェブアプリケーションがサポート予定のブラウザの対応状況を調査した上で、当該設定値の使用を検討してください。

³³ RFC7034 : 「HTTP Header Field X-Frame-Options」 <http://www.ietf.org/rfc/rfc7034.txt>

設定値	frame 要素および iframe 要素により表示できる範囲
DENY	すべてのウェブページにおいてフレーム内の表示を禁止
SAMEORIGIN	同一オリジンのウェブページのみフレーム内の表示を許可
ALLOW-FROM	指定したオリジンのウェブページのみフレーム内の表示を許可

9-(i)-b

☞ **処理を実行する直前のページで再度パスワードの入力を求め、実行ページでは、再度入力されたパスワードが正しい場合のみ処理を実行する。**

処理の実行前にパスワード認証を行うことにより、クリックジャッキングの脆弱性を解消できます。ただし、この方法は画面設計の仕様変更を要する対策であるため、画面設計の仕様変更をせず、実装の変更だけで対策をする必要がある場合には、9-(i)-a の対策を検討してください。

■ 保険的対策**9-(ii)**

☞ **重要な処理は、一連の操作をマウスのみで実行できないようにする。**

クリックジャッキング攻撃は、利用者を視覚的に騙して特定の操作をするように誘導します。そのため、利用者に複雑な操作をさせることは困難です。マウス操作のみで処理が実行されないように、キーボード操作などを挟むことで攻撃の成功率を下げることができます。

以上の対策により、クリックジャッキング攻撃に対する安全性の向上が期待できます。クリックジャッキングの脆弱性に関する情報については、次の資料も参考にしてください。

■ 対応する CWE

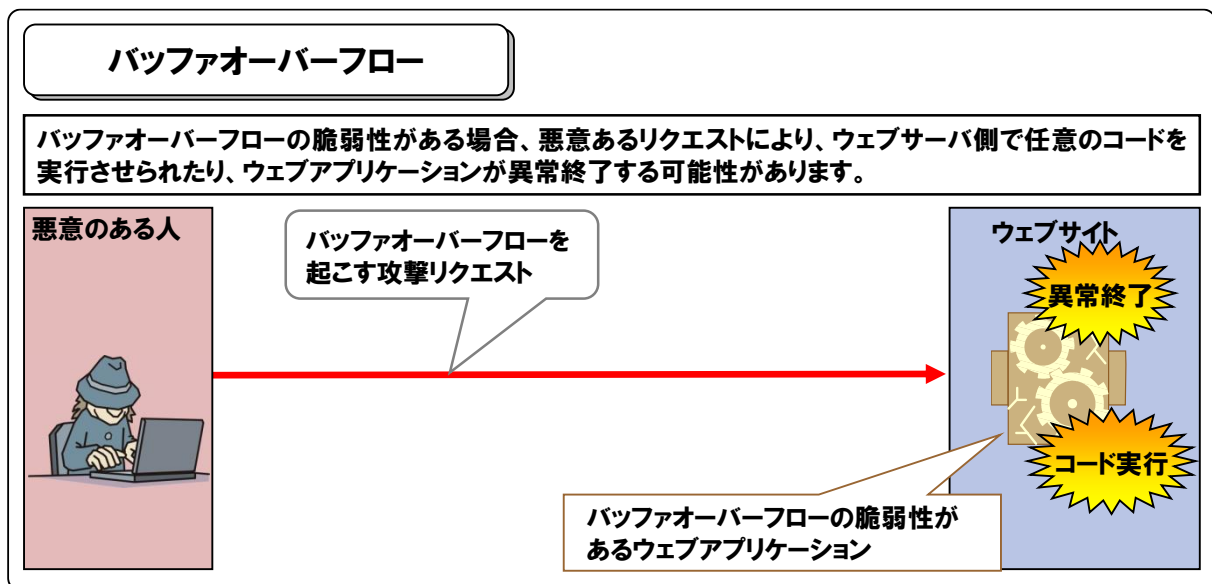
直接対応する CWE はありません。

■ 参考 URL

IPA: テクニカルウォッチ 『クリックジャッキング』に関するレポート
<https://www.ipa.go.jp/about/technicalwatch/20130326.html>

1.10 バッファオーバーフロー

ウェブアプリケーションを含む、あらゆるプログラムは、指示された処理を行うためにメモリ上に自身が使用する領域を確保します。プログラムが入力されたデータを適切に扱わない場合、プログラムが確保したメモリの領域を超えて領域外のメモリを上書きされ、意図しないコードを実行してしまう可能性があります。このような問題を「バッファオーバーフローの脆弱性」と呼び、この問題を悪用した攻撃を「バッファオーバーフロー攻撃」と呼びます。



■ 発生しうる脅威

バッファオーバーフローにより、発生しうる脅威は次のとおりです。

- プログラムの異常終了

- ・ 意図しないサービス停止

- 任意のコード実行

- ・ ウイルス、ワーム、ボット等への感染、バックドアの設置、他のシステムへの攻撃、重要情報の漏えい 等

■ 注意が必要なウェブサイトの特徴

バッファオーバーフローは C、C++、アセンブラなどの直接メモリを操作できる言語で記述されている場合に起こります。これらの言語を使って開発されたウェブアプリケーションを利用しているサイトは注意が必要です。

現在のウェブアプリケーションのほとんどは PHP や Perl、Java などの直接メモリを操作できない言語を使っており、バッファオーバーフローの脆弱性の影響を受ける可能性は低いといえますが、PHP や Perl、Java のライブラリの中にはバッファオーバーフローの脆弱性が存在していたものがあります。

■ 届出状況

バッファオーバーフローの脆弱性がウェブサイトに見つかったという届出を受けたことはありません。ソフトウェア製品においては、全体の数パーセント程度と多くはありませんが、継続的に届出を受けています。下記は、IPA が届出を受け、同脆弱性の対策が施されたソフトウェア製品の例です。

- ・複数のサイボウズ製品におけるバッファオーバーフローの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2014-000130>

- ・「Oracle Outside In」におけるバッファオーバーフローの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2013-000070>

- ・「茶釜 (ChaSen)」におけるバッファオーバーフローの脆弱性

<https://jvndb.jvn.jp/jvndb/JVNDB-2011-000099>

■ 根本的解決

10-(i)-a

☞ **直接メモリにアクセスできない言語で記述する。**

ウェブアプリケーションを直接メモリ操作できない言語で記述することで、バッファオーバーフローの脆弱性が作りこまれることを防げます。現在のウェブアプリケーションの多くは直接メモリを操作できない言語(PHP、Perl、Java など)で記述されており、これらの言語で作成されたウェブアプリケーションではバッファオーバーフローの問題は発生しません。

10-(i)-b

☞ **直接メモリにアクセスできる言語で記述する部分を最小限にする。**

プログラムの内部で C、C++、アセンブラなどの直接メモリにアクセスできる言語で記述された独自のプログラムを呼び出す場合は、その呼び出されるプログラムにバッファオーバーフローの脆弱性が存在する可能性があります。この直接メモリ操作可能な言語で記述作成された部分を最小限にし、その部分にバッファオーバーフローの脆弱性がないことを集中的に確認します。

10-(ii)

☞ **脆弱性が修正されたバージョンのライブラリを使用する。**

一般に流通しているライブラリを使用する場合、古いライブラリの中にはバッファオーバーフローの脆弱性が存在する場合がありますので、脆弱性が修正されたバージョンを使用してください。

■ 対応する CWE

CWE-119(バッファエラー)

<https://jvndb.jvn.jp/ja/cwe/CWE-119.html>

CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

<https://cwe.mitre.org/data/definitions/119.html>

1.11 アクセス制御や認可制御の欠落

ウェブサイトの中には、運営者のセキュリティに対する認識のなさから、不適切な設計で作成されたウェブサイトが運用されていることがあります。本節では、脆弱性関連情報として届出を受けた「アクセス制御」や「認可制御」等の機能欠落に伴う脆弱性についての対策を紹介します。

1.11.1 アクセス制御の欠落

■ 根本的解決

11-(i)

☞ **アクセス制御機能による防御措置が必要とされるウェブサイトには、パスワード等の秘密情報の入力を必要とする認証機能を設ける。**

ウェブサイトで非公開とされるべき情報を取り扱う場合や、利用者本人にのみデータの変更や編集を許可することを想定する場合等には、アクセス制御機能の実装が必要です。

しかし、たとえば、個人情報閲覧する機能にアクセスするにあたり、メールアドレスのみでログインできてしまうウェブサイトが、脆弱なウェブサイトとして届出を受けた例があります。

一般に、メールアドレスは他人にも知られ得る情報であり、そのような情報の入力だけで個人情報を閲覧できてしまうのは、アクセス制御機能が欠落していると言えます³⁴。

パスワード等(みだりに第三者に知らせてはならないものとして一般に考えられている情報)の入力を必要とするようにウェブアプリケーションを設計し、実装してください。

1.11.2 認可制御の欠落

■ 根本的解決

11-(ii)

☞ **認証機能に加えて認可制御の処理を実装し、ログイン中の利用者が他人になりすましてアクセスできないようにする。**

ウェブサイトアクセス制御機能を実装して、利用者本人にのみデータの閲覧や変更等の操作を許可する際、複数の利用者の存在を想定する場合には、どの利用者にもどの操作を許可するかを制御する、認可(Authorization)制御の実装が必要となる場合があります。

アクセス制御機能が装備されたウェブアプリケーションの典型的な実装では、ログインした利用者にはセッション ID を発行してセッション管理を行い、アクセスごとにセッション ID からセッション変数等を介し

³⁴ 「不正アクセス行為の禁止等に関する法律」では、第二条第二項で「識別符号」を定義しており、その第一号では、「当該アクセス管理者によってその内容のみだりに第三者に知らせてはならないものとされている符号」と定義しています。この定義に従うと、メールアドレスは識別符号に該当しないと解釈され、メールアドレスだけでログインする仕組みは、アクセス制御機能に該当しないと解される可能性があります。

不正アクセス行為の禁止等に関する法律: https://elaws.e-gov.go.jp/search/elawsSearch/elaws_search/lsg0500/detail?lawId=411AC0000000128

て利用者 ID を取得できるように構成されています。単純な機能のウェブアプリケーションであれば、その利用者 ID をキーとしてデータベースの検索や変更を行うように実装することができ、この場合は、利用者のデータベースエントリしか操作されることはないので、認可制御は結果的に実装されていると言えます。

しかし、ウェブサイトによっては、利用者 ID を URL や POST のパラメータに埋め込んでいる画面が存在することがあります。そのような外部から与えられる利用者 ID をキーにしてデータベースを操作する実装になっていると、ログイン中の利用者ならば他の利用者になりすまして操作できてしまうという脆弱性となります。

これは、認可制御が実装されていないために生じる脆弱性です。データベースを検索するための利用者 ID が、ログイン中の利用者 ID と一致しているかを常に確認するよう実装するか、または、利用者 ID を、外部から与えられるパラメータから取得しないで、セッション変数から取得するようにします。

また、他の例として、たとえば注文番号等をキーとしてデータベースの検索や変更を行う機能を持つウェブアプリケーションの場合、注文番号が URL や POST のパラメータで与えられる実装になっていると、ログイン中の利用者であれば、他人用に発行された注文番号を URL や POST のパラメータに指定することによって、他の利用者には閲覧できないはずの注文情報等を閲覧することができてしまう脆弱性が生じることがあります。

これも、認可制御が実装されていないために生じる脆弱性です。データベースを検索するための注文番号が、ログイン中の利用者には閲覧を許可された番号であるかどうかを常に確認するように実装してください。

2. ウェブサイトの安全性向上のための取り組み

ここでは、ウェブサイト全体の安全性を向上するための取り組みを掲載しています。前章「ウェブアプリケーションのセキュリティ実装」では、設計や実装レベルでの解決や対策を示しましたが、ここで取り上げている内容は、主に運用レベルでの解決や対策です。

2.1 ウェブサーバに関する対策

ウェブサイトを安全に運営するためには、ウェブアプリケーションのセキュリティ実装だけではなく、ウェブサーバのセキュリティ対策も考慮する必要があります。以下を参考に、管理しているウェブサーバの設定や運用に問題がないかを確認してください。

1)

👉 **OS やソフトウェアの脆弱性情報を継続的に入手し、脆弱性への対処を行う**

OS やソフトウェアの脆弱性をついた攻撃を受けると、たとえサーバへのアクセスに認証をかけていても、不正侵入されてしまう場合があります。脆弱性は日々発見されるので、OS やソフトウェアの開発者から提供される脆弱性情報を継続的に入手し、ソフトウェアの更新や問題の回避を行ってください。

2)

👉 **ウェブサーバをリモート操作する際の認証方法として、パスワード認証以外の方法を検討する**

サーバ管理の上で、ウェブサーバのリモート操作を許可する運用は一般的ですが、その際の認証方法にパスワード認証のみを利用している場合、総当たり攻撃等により、パスワード認証を突破されてしまう可能性があります。より高い安全性を確保するための方法として、暗号技術に基づく公開鍵認証等の利用を検討することをお勧めします。

3)

👉 **パスワード認証を利用する場合は、十分に複雑な文字列を設定する**

ウェブサーバへ接続する際のパスワードには、十分に複雑な文字列を設定してください。

4)

👉 **不要なサービスやアカウントを停止または削除する**

ウェブサイト運営に必要なないサービスがウェブサーバ上で稼働している場合、そのサービスに対する管理が十分でなく、脆弱性が存在するバージョンをそのまま利用している可能性等が考えられます。また、用途が明確でないユーザアカウントが存在している場合、そのアカウントに対する管理が十分でなく、不正利用される可能性が考えられます。必要の無いサービスやアカウントは停止または削除してください。

5) 公開を想定していないファイルを、ウェブ公開用のディレクトリ以下に置かない

ウェブ公開用のディレクトリに保管されているファイル群は、基本的に外部から閲覧することが可能です。公開ウェブページにファイルへのリンクが無くても、外部から直接指定することで閲覧されてしまいます。公開を想定していないファイルは、ウェブ公開用のディレクトリに保管しないようにしてください。

■ 参考 URL

IPA: 情報セキュリティ

<https://www.ipa.go.jp/security/>

JVN (Japan Vulnerability Notes)

<https://jvn.jp/>

JVN iPedia 脆弱性対策情報データベース

<https://jvndb.jvn.jp/>

IPA: テクニカルウォッチ「ウェブサイト改ざんの脅威と対策」

<https://www.ipa.go.jp/security/technicalwatch/20140829.html>

IPA: ウェブサイトの攻撃兆候検出ツール iLogScanner

<https://www.ipa.go.jp/security/vuln/iLogScanner/index.html>

2.2 DNS に関する対策

DNS はインターネット上でドメイン名を管理・運用するためのシステムです。DNS により、ドメイン名を指定するだけで、該当するウェブサイトへのアクセスやメールの送受信が可能になります。そのため、DNS に問題が発生すると、ウェブサイトや電子メール等のインターネットを利用するサービス全てに影響が及びます。以下を参考に、管理している DNS サーバの設定や運用に問題がないかを確認してください。

1) ドメイン名およびその DNS サーバの登録状況を調査し、必要に応じて対処を行う

ウェブサイトが利用しているドメイン名およびその DNS サーバについて、問題のある運用や設定は、悪意ある人によるドメイン名乗っ取りにつながる可能性があります。ドメイン名の乗っ取りを行われた場合、利用者が本物のウェブサイトの URL を指定しても、そのドメイン名を乗っ取った人が用意したウェブサイトに接続してしまいます。ドメイン名およびその DNS サーバについて、登録状況を確認し、必要に応じて対処を行ってください。

2) DNS ソフトウェアの更新や設定を見直す

DNS を狙った攻撃として、キャッシュポイズニング攻撃やリフレクター攻撃、水責め攻撃が存在します。DNS ソフトウェアの設定不備や古いバージョンが原因となり、キャッシュポイズニング攻撃の場合は被害を受けやすくなり、リフレクター攻撃や水責め攻撃の場合は踏み台に悪用される可能性が高ま

ります。これらの攻撃は、DNS の仕組みを悪用しているため、完全に防ぐことは困難ですが、DNS ソフトウェアの更新や設定変更をすることで、影響を緩和することができます。

また、DNS ソフトウェア自体に脆弱性が見つかることもあります。脆弱性をついた攻撃を受けると、DNS サーバが異常終了する場合があります。ソフトウェアの開発者から提供される脆弱性情報を継続的に入手し、ソフトウェアの更新や問題の回避を行ってください。

DNS サーバの運用を外部に委託している場合は、その委託先に対処を依頼する必要があります。DNS に関する攻撃や対策の詳細については、次の資料を参考にしてください。

■ 参考 URL

IPA: ドメイン名の登録と DNS サーバの設定に関する注意喚起

https://www.ipa.go.jp/security/vuln/20050627_dns.html

JPRS: 登録情報の不正書き換えによるドメイン名ハイジャックとその対策について

<https://jprs.jp/tech/security/2014-11-05-unauthorized-update-of-registration-information.html>

JPRS: キャッシュポイズニング攻撃対策: 権威 DNS サーバ運用者向け—基本対策編

<https://jprs.jp/tech/security/2014-05-30-poisoning-countermeasure-auth-1.pdf>

JPRS: キャッシュポイズニング攻撃対策: キャッシュ DNS サーバ運用者向け—基本対策編

<https://jprs.jp/tech/security/2014-04-30-poisoning-countermeasure-resolver-1.pdf>

JPCERT/CC: オープンリゾルバ確認サイト公開のお知らせ

<https://www.jpCERT.or.jp/pr/2013/pr130002.html>

JPRS: Bot 経由で DNS サーバを広く薄く攻撃 ~DNS 水責め攻撃の概要と対策~

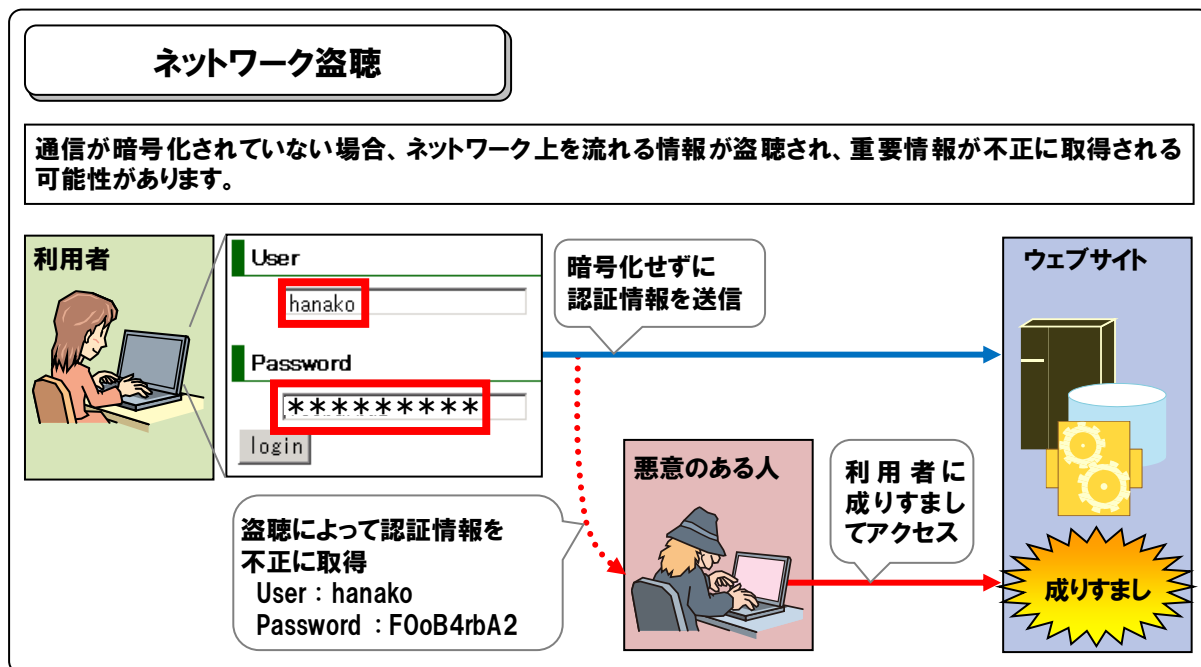
<https://jprs.jp/related-info/guide/021.pdf>

JPRS: DNS 関連技術情報

<https://jprs.jp/tech/>

2.3 ネットワーク盗聴への対策

ウェブサイトと利用者間で交わされる情報は、ネットワークの盗聴によって不正に取得される可能性があります。通信や情報が暗号化されていない場合、盗聴によって取得された情報が悪用され、なりすまし等につながる可能性があります。



ネットワーク盗聴はウェブサイトと利用者との経路上で行われるため、この行為自体をウェブサイト側の運用や設定のみで防ぐことは困難です。しかし、通信経路を暗号化すれば、盗聴を受けても重要情報が不正に取得されることを防止できます。特に認証情報や個人情報を扱うウェブサイトでは、ネットワーク盗聴への対策として、次の内容を検討してください。

1)

👉 重要な情報を取り扱うウェブページでは、通信経路を暗号化する

ウェブサイトで通信を暗号化する手段として、SSL(Secure Socket Layer)や TLS(Transport Layer Security)を用いた HTTPS 通信があります。パスワードでログインするページや、個人情報を登録するページ、また、秘密にするべき情報を表示する画面のページは、https:// で始まる URL として、通信経路を暗号化することをお勧めします。

暗号化したい情報を入力させる画面では、送信先の URL を https:// とするだけでなく、入力画面も https:// の URL としておく必要があります。そうしなければ、入力画面が盗聴者に改ざんされている可能性があり、利用者が改ざんに気づかずに入力すれば、差し替えられた別のサイトに送信されてしまったり、暗号化されずに送信されて盗聴される危険があるからです。利用者は入力画面が https:// になっていることを確認してから入力しますので、ウェブサイト運営者は、そのような確認ができるようにしてください。

利用者がこの確認を怠り、http:// の画面でパスワード等を入力してしまう可能性があるため、それを防止する「HSTS(HTTP Strict Transport Security)」の機能が、2012 年に RFC 6797 で規定されました。最新の主要なウェブブラウザはこれに対応しています。この機能を有効にするにはウェブサイト側

で設定が必要であり、ウェブサーバのレスポンスヘッダに「Strict-Transport-Security」を含めるように設定します。そうすることで、一度そのサイトを訪れた利用者のブラウザは、それ以降、そのサイトには https:// でしか接続しないようになります。

HSTS の機能を利用するには、サイト全体を https:// の画面とするよう設計しなければなりません。パスワードの入力・送信だけでなく、ログイン中のセッションの全部で HTTPS 通信が使われるようにします。セッション管理に用いるセッション ID を格納する cookie には、secure 属性を加えるようにします (4-(iii)参照)。

2)

☞ 利用者へ通知する重要情報は、メールで送らず、暗号化された https://のページに表示する

ウェブサイトの運営によっては、ウェブサイトの利用者に、個人情報やパスワード等の重要情報を通知する場合があります。ここで、ネットワークを経由して情報を送信する場合には、盗聴対策として通信の暗号化か、重要情報の暗号化が必要になります。暗号化が必要な情報を利用者に通知する場合は、HTTPS 通信を利用し、ウェブページに表示することをお勧めします。

メールを利用する場合には、メール本文の暗号化として、S/MIME (Secure / Multipurpose Internet Mail Extensions) や PGP (Pretty Good Privacy) 等の技術がありますが、利用者側に暗号化環境やプライベートキー (秘密鍵) が必要となるため、現実的ではないかもしれません。

3)

☞ ウェブサイト運営者がメールで受け取る重要情報を暗号化する

ウェブページに入力された個人情報等の重要情報を、ウェブアプリケーションに実装されたメール通知機能を利用して、ウェブサイト運営者がメールで受け取る場合は、S/MIME や PGP 等を利用してメールを暗号化するようにしてください。S/MIME や PGP を利用できない場合には、その他の方法でメール本文を暗号化するようにします。

なお、盗聴対策として、メールサーバ間の通信の暗号化 (SMTP over SSL) やメールサーバとウェブサイト運営者との通信の暗号化 (POP/IMAP over SSL) 等も考えられますが、ネットワーク構成によっては、途中経路が暗号化されない可能性があるため、安全とは言えません。

■ 参考 URL

IPA: 電子メールのセキュリティ 電子メールの安全性を高める技術の利用法

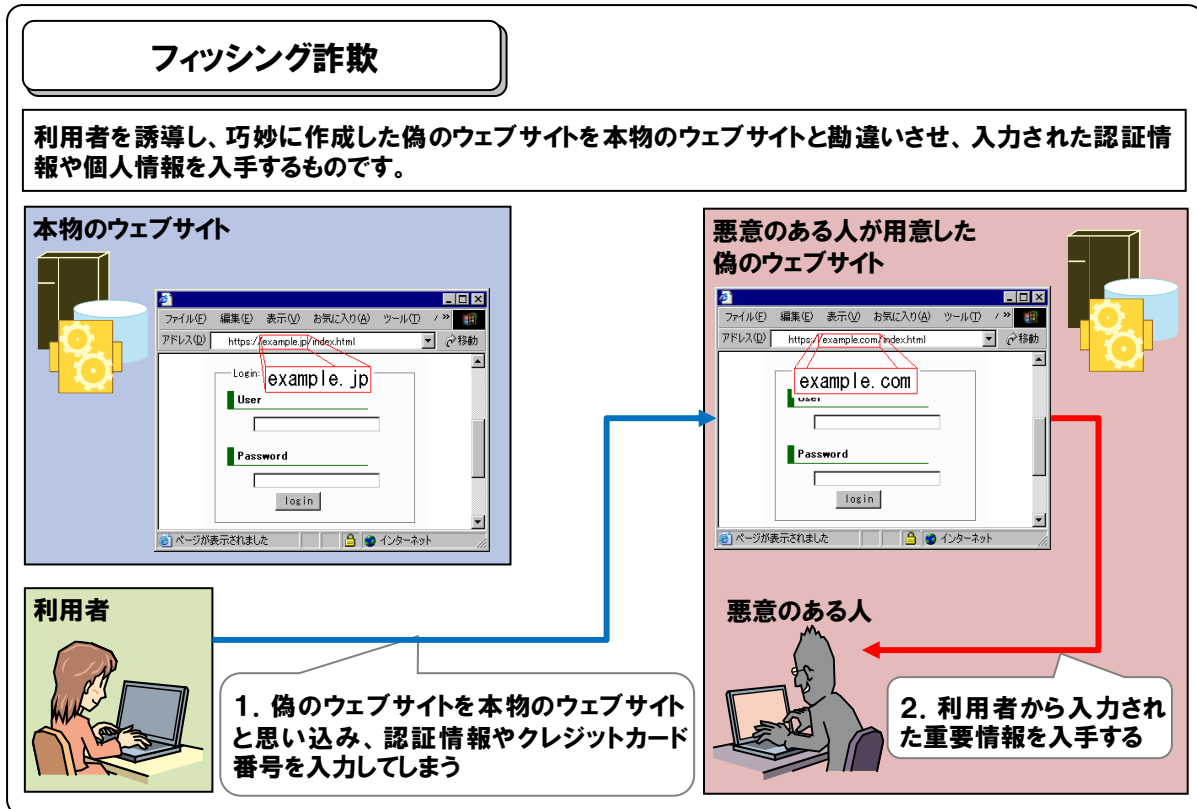
<https://www.ipa.go.jp/files/000029547.pdf>

IPA: 電子メールのセキュリティ「S/MIME を利用した暗号化と電子署名」

https://www.ipa.go.jp/security/fy12/contents/smime/email_sec.html

2.4 フィッシング詐欺を助長しないための対策

フィッシング詐欺とは、悪意のある人が、金融サイトやショッピングサイト等を装った偽のウェブサイトを作成して、利用者を巧みにそこへ誘導して、利用者の認証情報やクレジットカード番号等を不正に取得するものです。フィッシング詐欺の回避には、利用者側の注意が必要ですが、ウェブサイトの運用によっては、利用者の注意を妨げ、結果としてフィッシング詐欺を助長してしまう場合があります。



ウェブサイト利用者がフィッシング詐欺の被害に遭わないためには、利用者自身がアクセスしたウェブサイトを注意深く確認し、本物のウェブサイトかどうかを見極める必要があります。利用者が本物のウェブサイトであることを正しく確認できるよう、ウェブサイト運営者は次の点を検討してください。

1)

👉 **EV SSL 証明書を取得し、サイトの運営者が誰であることを証明する**

サーバ証明書は、SSL の暗号化通信を正しく実現するために必要なものですが、同時に、ウェブサイトの運営者が誰であることを証明する目的でも利用することができます。EV SSL 証明書を用いると HTTPS 通信でアクセスした際、ブラウザのアドレスバーに組織名が緑色で表示されます。利用者は、パスワードやクレジットカード番号等を入力する画面で、閲覧中のサイトの運営者が誰であることを確認できるようになります。

2)

👉 **フレームを利用する場合、子フレームの URL を外部パラメータから生成しないように実装する**

フレームを利用しているウェブページで、子フレームの URL を外部パラメータから生成する実装は、フィッシング詐欺に悪用される危険性があります。そのパラメータに任意の URL を指定したリンクを仕

掛けられた場合、そのリンクをアクセスした利用者は、本物サイトの親フレーム内に、偽サイトのウェブページを子フレームとして埋め込まれた画面を閲覧することになります。表示上のドメインは本物であるため、利用者が子フレームを偽サイトと見分けることは困難です。

3)

☞ **利用者がログイン後に移動するページをリダイレクト機能で動的に実装しているウェブサイトについて、リダイレクト先の URL として使用されるパラメータの値には、自サイトのドメインのみを許可するようにする**

ウェブサイトの中には、利用者がログイン後に閲覧可能な URL にログアウトした状態でアクセスした場合、その URL 情報をパラメータの値等で保持してログイン画面を表示し、ログイン成功後に、そのパラメータの値を利用して改めてリダイレクトするものがあります。しかし、この「リダイレクト先の URL として使用されるパラメータの値」に制限が無い場合、このパラメータに罠の URL を指定されることにより、フィッシング詐欺に悪用される可能性があります。

この罠にかかった場合、注意深い利用者は、最初に表示されるログイン画面が正規のウェブサイトであるかどうかは確認するはずですが、そして、このログイン画面が正規のウェブサイトであるため、安心してログインします。しかし、ログイン後にリダイレクトされるページが偽のウェブサイトであることまで、注意を継続することはできないかもしれません。たとえば、リダイレクト先の罠ページが、正規のウェブサイトのログイン画面とそっくりで、「ログイン失敗、再入力を」というメッセージがあった場合、「あれ、パスワードを間違えたかな？」と大きな疑いを抱かずに認証情報を入力してしまうことが予想されます。

リダイレクト先の URL として使用されるパラメータについては、任意の URL を許可せず、自サイトのドメインのみを許可するようにしてください。かつ、この対策は、リダイレクトを利用している全てのウェブページに対して漏れなく実施してください。

フィッシング詐欺を助長しないための対策について、下記の資料も参考にしてください。

■ 参考 URL

IPA: PKI 関連技術解説 「認証局と電子証明書」

<https://www.ipa.go.jp/security/pki/031.html>

産業技術総合研究所: 安全な Web サイト利用の鉄則

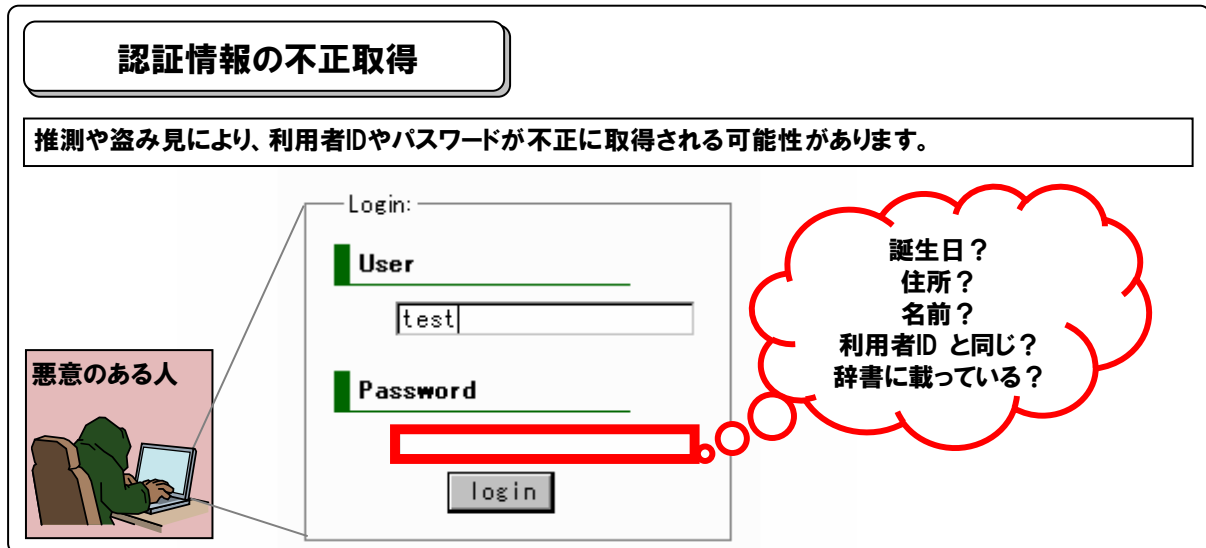
<https://www.rcis.aist.go.jp/special/websafety2007/>

フィッシング対策協議会: フィッシング対策ガイドラインの改訂について

https://www.antiphishing.jp/report/guideline/antiphishing_guideline2014.html

2.5 パスワードに関する対策

ウェブサイトにおける利用者の認証は、ユーザ ID とパスワードを用いる方法が一般的です。しかし、パスワードの運用やウェブページ上のパスワードの取り扱い方法に問題がある場合、利用者の認証情報が悪意ある人に不正取得される危険性が高まります。



認証情報の不正取得の手段の一つに、ユーザ ID やパスワードの「推測」があります。これは、推測されやすい単純なパスワードで運用している場合に悪用される手段ですが、ウェブページの表示方法によっては、さらに推測のヒントを与えてしまう場合があります。利用者の認証を行うウェブサイトでは、次の内容に注意してください。

1)

👉 **初期パスワードは、推測が困難な文字列で発行する**

初期パスワードは、暗号論的擬似乱数生成器を利用して規則性をなくし、可能であれば英数字や記号を含めた長い文字列で発行してください。パスワード発行に規則性がある場合、調査のためのテストユーザを複数登録され、その際に発行されたパスワードから規則性を導き出されてしまうかもしれません。利用者によっては、初期パスワードを変更せずに継続して利用することも考えられるため、初期パスワードが推測しやすい仕様は避けるべきです。

2)

👉 **パスワードの変更には、現行パスワードの入力を求める**

パスワードの変更には、必ず現行パスワードの入力を求めるようにしてください。

3)

👉 **入力後の応答メッセージが認証情報の推測のヒントとならない工夫をする**

認証画面で利用者が入力を誤った際、遷移後の画面で「パスワードが間違っています」というエラーメッセージを表示するものは、「ユーザ ID は正しく、パスワードが間違っている」ということを示していることとなります。このような表示内容は、登録されているユーザ ID の割り出しを容易にしてしまうため、

お勧めできません。入力後の応答メッセージには、「ユーザ ID もしくはパスワードが違います」というような表示を用い、認証情報の推測のヒントを与えない工夫をしてください。

4)

☞ 入力フィールドでは、パスワードは伏せ字で表示されるようにする

利用者にパスワードを入力させるときは、ブラウザに備わっているパスワード専用の入力フィールドを用いるようにしてください。これにより、入力したパスワード文字列は伏せ字(アスタリスク “*”)で表示されます³⁵。

5)

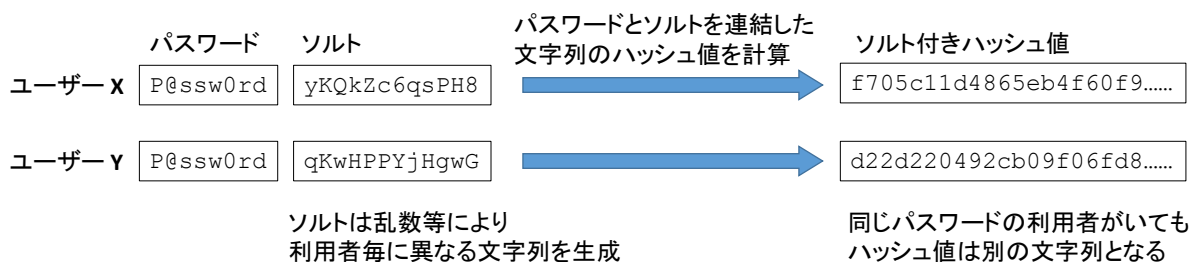
☞ パスワードをサーバ内で保管する際は、平文ではなくソルト付きハッシュ値の形で保管する

ウェブサイト内で保管したパスワードが SQL インジェクション攻撃などにより外部に漏洩した場合、パスワードリスト攻撃等の不正ログインに悪用される可能性が高くなります。このため、万一パスワード情報が漏洩した場合でも直ちに悪用されることのないように、パスワードを保護した形で保管します。この目的のためには、パスワードのハッシュ値(暗号論的ハッシュ関数により計算)の形で保管することが一般的に行われています。

ハッシュ値から平文字列を復元することは一般的には困難ですが、弱いパスワードの場合は辞書攻撃により元のパスワードを復元できてしまいますし、複数の同じハッシュ値を探すことで、同一の弱いパスワードをつけているアカウントを簡単に見つけることができます。また、辞書攻撃で復元できないパスワードであっても、十分に長いパスワードでないと、総当たり攻撃により、十分な時間をかければ元のパスワードを復元できてしまう問題があります。そして、それを高速に実現するレインボーテーブルという技法が開発されています。

これらに対し、保管するハッシュ値を、パスワードにソルトと呼ばれるユーザ毎に異なる文字列をつけてからハッシュ値を求めたものとするにより、見かけのパスワードを長くして、レインボーテーブル攻撃を回避できますし、前記の同一のパスワードの発見も回避できます。したがって、このようなソルト付きハッシュ値の形でパスワードを保管することをお勧めします。

さらに、それでもなお、弱いパスワードや十分に長くないパスワードは、十分な時間をかければ復元されてしまうので、復元にかかる計算時間を長くするために、あえて計算の遅いハッシュ関数を使う技法があります。そのような遅いハッシュ関数を実現する一つの方法として、ハッシュ値をさらにハッシュ関数にかける計算を繰り返し行う方法があり、この方法はストレッチングと呼ばれています。



³⁵ これにより、入力した文字の目視確認ができなくなるので、長いパスワードを入力するには不便ですが、最近の一部 OS では、利用者の意思により一時的に入力した文字を画面に表示する機能を持つものがあります。

■ 参考 URL

IPA: パスワードリスト攻撃による不正ログイン防止に向けた呼びかけ(サービス利用者向けの解説)

<https://www.ipa.go.jp/about/press/20140917.html>

PHP.net: パスワードのハッシュ - Manual

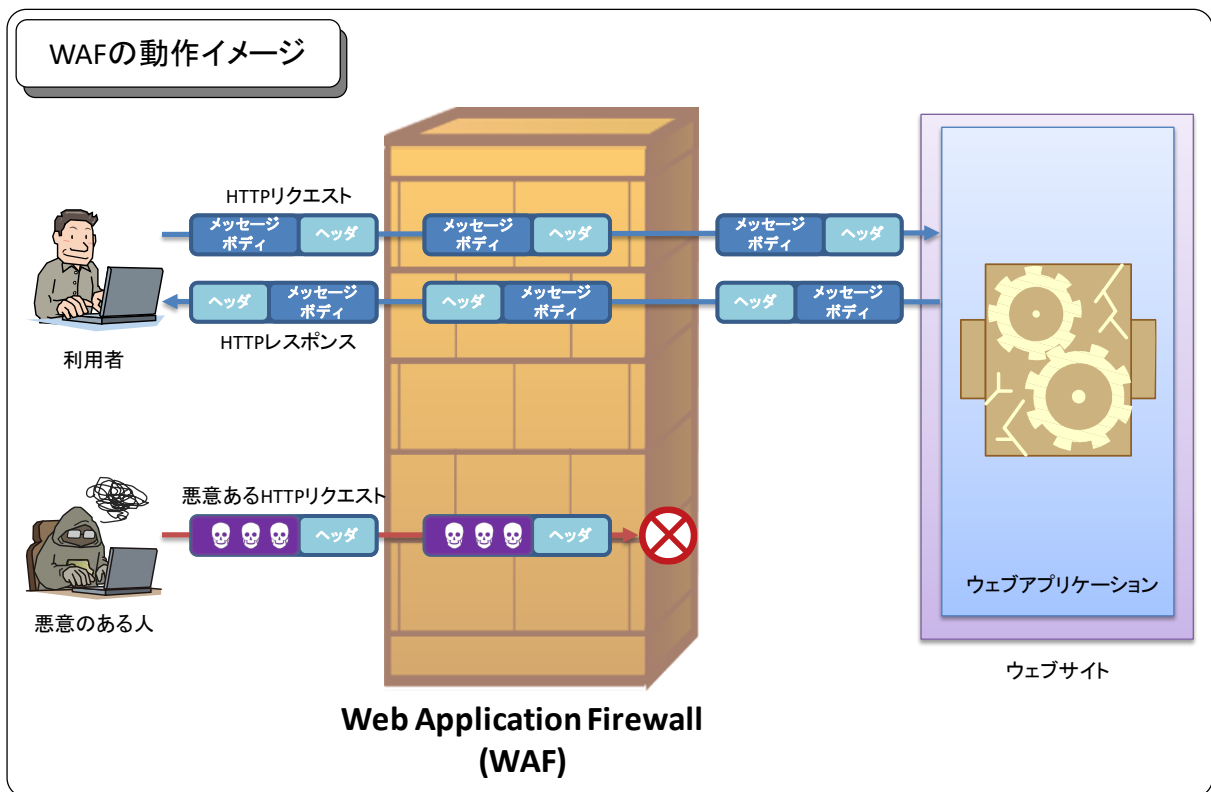
<https://www.php.net/manual/ja/faq.passwords.php>

2.6 WAF によるウェブアプリケーションの保護

ウェブアプリケーションの安全を確保するには、脆弱性を作り込まないことや、脆弱性が発見されたら早期に該当箇所を修正することが重要です。一方、そのようなウェブアプリケーションの実装面での対策とは別に、ウェブアプリケーションの脆弱性を悪用した攻撃からウェブアプリケーションを保護する運用面での対策として、WAF (Web Application Firewall) の使用があります。

WAF は、ウェブアプリケーションを含むウェブサイトと利用者の間で交わされる HTTP (HTTPS 通信を含む³⁶) を検査し、攻撃等の不正な通信を自動的に遮断するソフトウェア、もしくはハードウェアです。WAF を使用することで以下の効果を期待できます。

- 脆弱性を悪用した攻撃からウェブアプリケーションを防御する
- 脆弱性を悪用した攻撃を検出する
- 複数のウェブアプリケーションへの攻撃をまとめて防御する



WAF の動作イメージ

ウェブアプリケーションの開発状況や運用状況によっては、ウェブアプリケーションの実装面での対策よりも、WAF の使用が有効な場合があります。

³⁶ WAF によっては、HTTPS 通信を検査できないため注意してください。

■ WAF の使用が有効な状況 ～ウェブアプリケーションの改修が困難な状況～

開発者は、本書 1 章「ウェブアプリケーションのセキュリティ実装」に基づいて、脆弱性を作り込まないようにウェブアプリケーションを実装すべきです。しかし、開発が完了した後でウェブアプリケーションに脆弱性が発見される場合もあります。その場合、早期に脆弱性を修正すべきですが、ウェブアプリケーションの改修が困難な状況が考えられます。このような状況において、WAF は攻撃による影響を低減する対策としてウェブアプリケーションを保護することができます。たとえば、下記のような状況です。

1) 開発者にウェブアプリケーションの改修を依頼できない状況

ウェブアプリケーションに脆弱性が発見された場合、開発者に直接脆弱性の修正を依頼できないことがあります。

企業や組織がウェブアプリケーションを開発する際、他社に開発を依頼することがあります。仮にこのウェブアプリケーションに脆弱性が発見された場合に、開発企業に脆弱性の修正を依頼できない事態(例: 開発事業から撤退している)が生じ得ます。

開発企業にウェブアプリケーションの改修を依頼せずとも、他の企業に改修を依頼することもできます。しかし、改修費用が高くなり予算内で改修できない事態に陥る可能性があります。

2) 改修できないウェブアプリケーションに脆弱性が発見された状況

商用製品やオープンソースソフトウェアを使用してウェブサイトを構築した場合、該当ソフトウェアの改修に直接関与できず、脆弱性を修正できないことがあります。

近年、ブログや Wiki に代表されるウェブアプリケーションが商用製品やオープンソースソフトウェアとして提供されています。これらのソフトウェアを利用することで、ウェブアプリケーションを独自開発することなく、ウェブアプリケーションを利用できます。

上記のようなソフトウェアに脆弱性が発見された場合、該当ソフトウェアの開発元が脆弱性を修正したバージョン、または修正パッチを提供しない限り、脆弱性を修正できません。該当ソフトウェアのサポート期間が終了していた場合、脆弱性が修正されない可能性もあります。

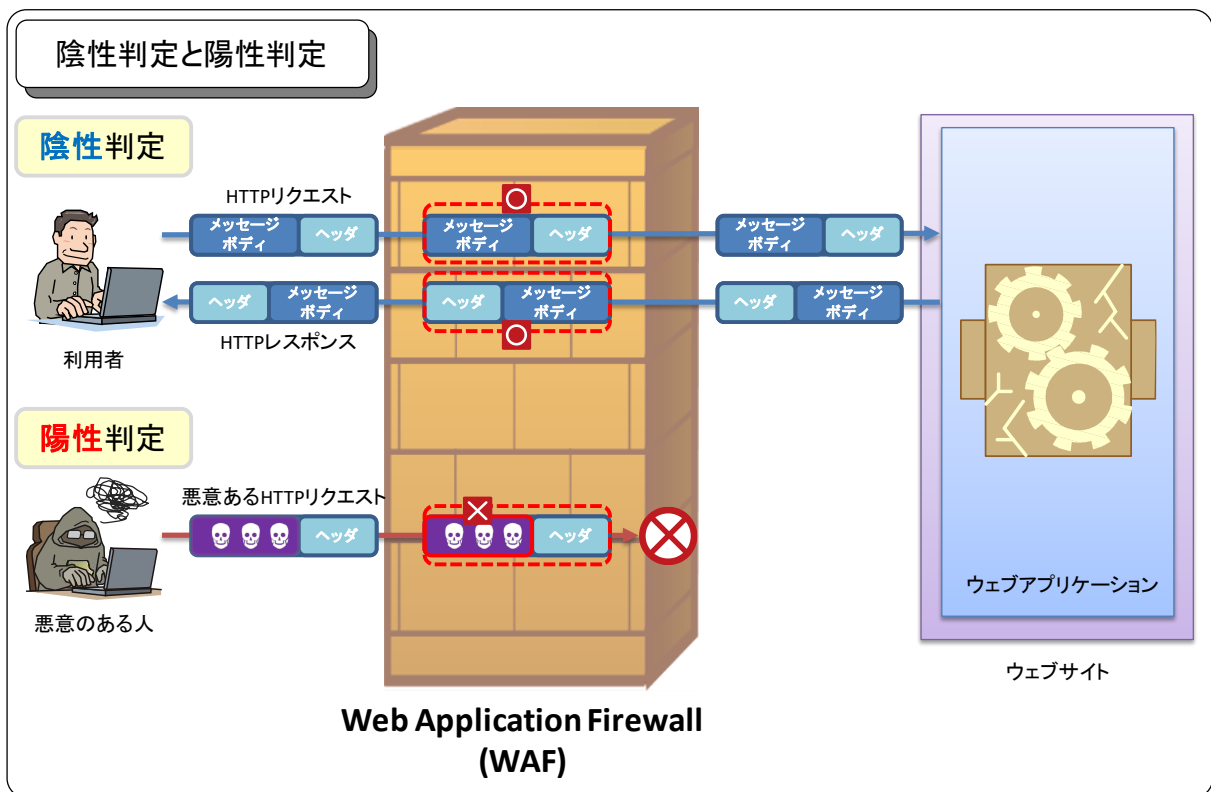
オープンソースソフトウェアの場合、利用者自身が脆弱性を確認し修正することもできます。ただし、自組織に脆弱性を修正できる技術者がいない場合もあるでしょう。

■ WAF における HTTP 通信の検査

WAF は WAF を導入したウェブサイト運営者が設定する検出パターンに基づいて、ウェブサイトと利用者の間で交わされる HTTP 通信内の HTTP リクエスト、HTTP レスポンスそれぞれの中身を機械的に検査します。WAF は、検査の結果から HTTP 通信がウェブサイト、利用者にとって「悪いもの」かどうかを判定します。検出パターンには、「ウェブアプリケーションの脆弱性を悪用する攻撃に含まれる可能性の高い文字列」や「ウェブアプリケーション仕様で定義されているパラメータの型、値」³⁷といったものを定義します。

WAF が HTTP 通信を「正常である」と判定した場合（陰性判定）、検査した HTTP 通信を利用者またはウェブサイトそのまま送信します。一方、WAF が HTTP 通信を「悪質である」と判定した場合（陽性判定）には、WAF は検査した HTTP 通信を送信せずに設定された処理（管理者への警告、該当通信の遮断等）を実行します。

WAF は HTTP 通信を機械的に検査しているため、人の目で見ると間違った判断となる陰性判定、陽性判定（以降、判定エラー）が生じる場合があります。



陰性判定と陽性判定

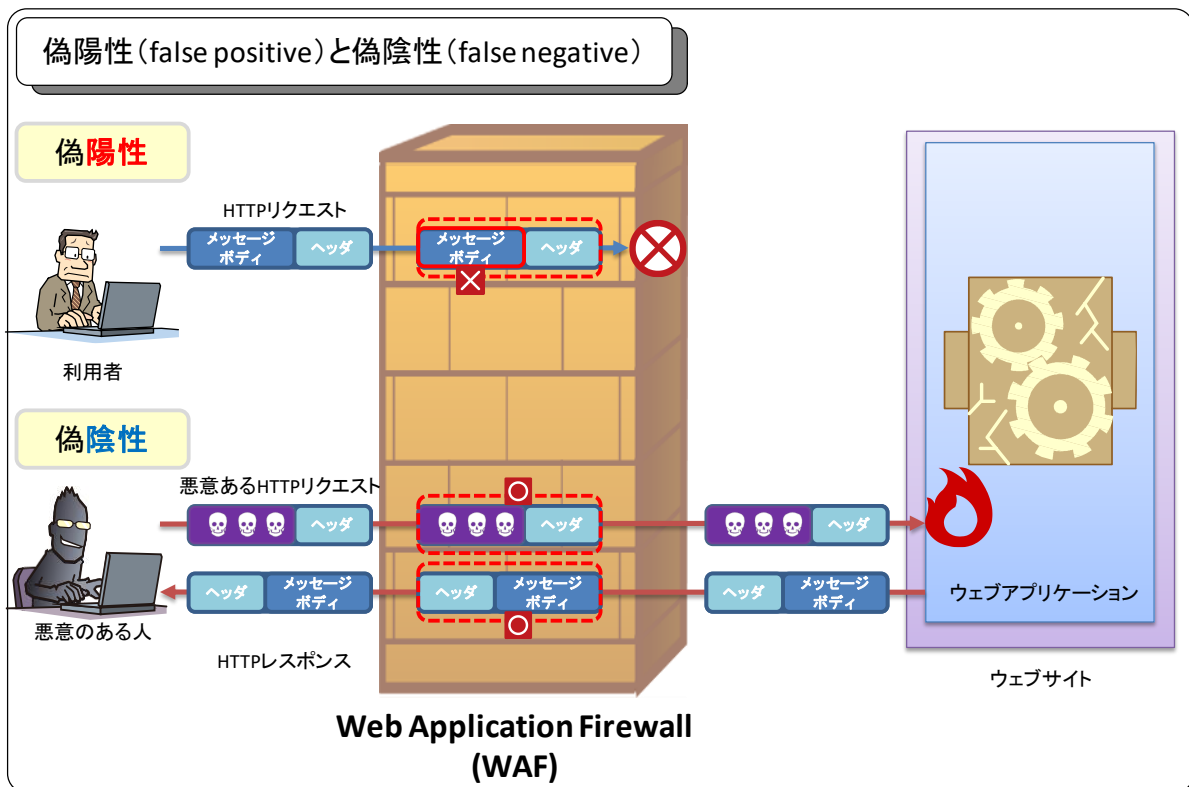
³⁷ たとえば、あるウェブアプリケーションが id というパラメータを受け取るものとします。このウェブアプリケーションが id パラメータの値として数値を期待する場合、数値以外の値（例：文字列「example」）は、このウェブアプリケーションにとって正しい値ではありません。このウェブアプリケーションを WAF の防御対象とする場合、「id パラメータは数値のみ」という検出パターンを定義できます。

■ HTTP 通信の検査における判定エラー

HTTP 通信の中身によっては、判定エラーが生じる場合があります。判定エラーには偽陽性・偽陰性の 2 種類があります。

偽陽性とは、本来「正常である」にもかかわらず、「悪質である」と判定されるエラーです。英語では一般的に false positive と呼ばれます。偽陰性とは、本来「悪質である」にもかかわらず、「正常である」と判定されるエラーです。英語では一般的に false negative と呼ばれます。

WAF を使用する場合、偽陽性 (false positive)、偽陰性 (false negative) の判定が生じる可能性を考慮する必要があります。



偽陽性 (false positive) と偽陰性 (false negative)

■ WAF における偽陽性と偽陰性

1) 偽陽性 (false positive)

【原因】

WAF における偽陽性は、利用者と保護対象ウェブアプリケーションの間で交わされる HTTP 通信にあわせた検出パターンを定義していないことから生じます。

【影響】

正当な通信が遮断されることで、ウェブサイトの可用性が損なわれる。

【例】

安易な検出パターンを定義することで、利用者の正当な HTTP 通信を WAF で遮断してしまう

クロスサイト・スクリプティングの脆弱性を悪用する攻撃からウェブサイト利用者を防御する例を考えます。この攻撃を WAF で防御するため、HTML の特殊文字「<」、「>」を検出パターンとして定義し、検出した場合は遮断するようにしたと仮定します。この場合、ウェブサイト利用者がウェブアプリケーション上で「<」、「>」を含む数式を入力しただけで、サイト利用者の正当な HTTP 通信が WAF で遮断されてしまう可能性があります。

一般的な WAF の検出パターンでは、この例のような極端に安易なものは含まれませんが、WAF の原理上、偽陽性の問題は生じる可能性があります。

2) 偽陰性 (false negative)

【原因】

WAF における偽陰性は、以下の 2 つの要因から生じます。

- a) 不正な HTTP 通信を判定する検出パターンを定義できない
- b) 偽陽性の生じる可能性を最小限にするため、検出パターンを減らした

【影響】

ウェブアプリケーションの脆弱性を悪用する攻撃からウェブアプリケーションを防御できない。

【例】

WAF とウェブアプリケーションでの動作の差異により、悪意ある HTTP 通信を WAF で検出できない

HTTP リクエストにおいて、クエリストリングとメッセージボディ、Cookie に同じ名前のパラメータが複数存在した場合、ウェブアプリケーションの言語とミドルウェア、ウェブサーバの実装によって、そのパラメータの取り扱いに差異が生じます。この差異を悪用して³⁸、同じ名前のパラメータに脆弱性を悪用する攻撃の文字列を分割して、HTTP リクエストを送信することで、WAF が脆弱性を悪用する攻撃を検出できない事象が生じる場合が報告されています。

³⁸ 同じ名前のパラメータが複数存在した場合の動作の差異を悪用した手法は、HTTP Parameter Pollution (HPP)と呼ばれています。

資料“HTTP Parameter Pollution³⁹”は、オープンソースソフトウェアの ModSecurity において、「select 1,2,3」という検出パターンを定義していた場合、以下の HTTP リクエストが送信されると、ModSecurity がこの HTTP リクエストを攻撃として検出できないことを指摘しています⁴⁰。

```
index.aspx?page=select 1&page=2,3 from table where id=1
```

プロトコルの仕様のうち、RFC 等の文書が明確に定義していない部分は、言語とミドルウェア、ウェブサーバの開発者が独自の解釈に基づいて、プロトコルを実装します。このときの解釈の違いにより、各ソフトウェアにおいて動作の差異が生じます。この差異は WAF においても同様です。この差異を悪用されると、脆弱性を悪用した攻撃を WAF で検出できない場合があります。

■ WAF の導入検討における留意点

WAF を導入するに際して、偽陽性と偽陰性の判定が生じる可能性を低くするためには、まず、WAF が検出パターンに合致する HTTP 通信を検出しても HTTP 通信を遮断しないように設定し、HTTP 通信を監視するだけのテスト期間を設けます。このテスト期間に WAF の保護対象ウェブアプリケーションを実際に使用して正当な HTTP 通信が遮断されないか、また保護対象ウェブアプリケーションにあわせて WAF の検出パターンを適切に設定しているか、といった WAF の動作確認を実施します。この動作確認を実施するには、保護対象ウェブアプリケーションの理解や HTTP 通信に関連したプロトコルの専門的知識が要求され、かつ十分な作業工数が必要です。そのため、外部の専門家に WAF の導入を依頼することも検討してください。

WAF によるウェブアプリケーションの保護について、下記の資料も参考にしてください。

■ 参考 URL

IPA: Web Application Firewall 読本
<https://www.ipa.go.jp/security/vuln/waf.html>

³⁹ Luca Carettoni, Stefano diPaola, “HTTP Parameter Pollution”, OWASP AppSec Europe 2009
https://owasp.org/www-pdf-archive/AppsecEU09_CarettoniDiPaola_v0.8.pdf

⁴⁰ 2009 年 10 月現在の最新バージョン ModSecurity v2.5.10 + Core Rule Set v2.0.2 では HPP の検出パターンが定義されています。

2.7 携帯ウェブ向けのサイトにおける注意点

本書で説明している一連の対策は、そのウェブサイトが PC 向けか携帯ウェブ⁴¹向けかに関わらず必要なものです。しかし、携帯ウェブ向けのサイトを作る際には、機能の制限から、PC 向けとは違ったサイト設計をする必要が生じることがあります。本節では、そのような携帯ウェブ向けのサイトで起きやすい問題と、注意すべき点を示します。携帯ウェブ向けのサイトを作る際は、本節のトピックについても考慮したうえで、場合によってはウェブサイトの設計を変更することも検討してください。

2.7.1 セッション管理に関する注意点

2009 年 5 月までは、一部の携帯電話事業者(以下、「キャリア」とする)のすべての機種において、携帯ウェブのブラウザが、HTTP の基本的な機能である Cookie と Referer に非対応でした。そのため、やむを得ずそれに合わせてウェブサイトを開発する必要がありました。

しかし 2009 年 5 月以降、そのキャリアにおいても、Cookie と Referer の機能に対応する機種が混在するようになりました。

Cookie 機能がない機種では、セッション管理のためセッション ID を URL に格納せざるを得ませんでした。1.4 節の根本的解決 4-(ii)に示したように、一般的には、セッション ID を URL パラメータに格納していると、利用者のブラウザが、Referer 送信機能によって、セッション ID の含まれた URL をリンク先のサイトへ送信してしまい、セッション・ハイジャック攻撃につながる危険があります。そのため、携帯ウェブ向けのサイトでは、外部サイトへのリンクを作らないようにするか、外部サイトへのリンクを作る場合であっても、URL にセッション ID を含まないページを間に挟むようにする等の対策が取られているようです。しかし、その場合でも、利用者が自ら URL を公開したこと等が原因となって、その URL のページが検索エンジンに登録されることによる個人情報漏洩事故が発生していることから、根本的な解決にはなっていません。

可能な限りこのような実装は避けるべきであり、Cookie 機能に非対応のキャリアにだけ上記の回避策をとり、それ以外のキャリアに対しては Cookie 機能を用いて通常の一般的な PC 向けウェブサイトと同様に実装するのが適切でした。しかし、2009 年 5 月以降、同じキャリアでも Cookie 機能に非対応の機種と対応する機種が混在するようになったため、キャリア単位ではなく、機種ごとに実装方法を分けるべきと言えます。

このような変化の中で、古くから存在する携帯ウェブ独自のノウハウを適用してウェブサイトを作ることは、ウェブサイトの安全を損なう原因になることがあります。古いノウハウを見直していくことが必要です。

2.7.2 クロスサイト・スクリプティングに関する注意点

2009 年までに発売の携帯電話では、JavaScript 非対応の機種が大半でしたが、2009 年以降、JavaScript をはじめ XMLHttpRequest 機能等にも対応した機種が発売され始めており、PC に近づく形で高機能化が進んでいます。

⁴¹ 本節における携帯ウェブとは、日本のキャリアが提供するゲートウェイを介したウェブ接続サービス(「i モード」や「EZweb」等)を示します。

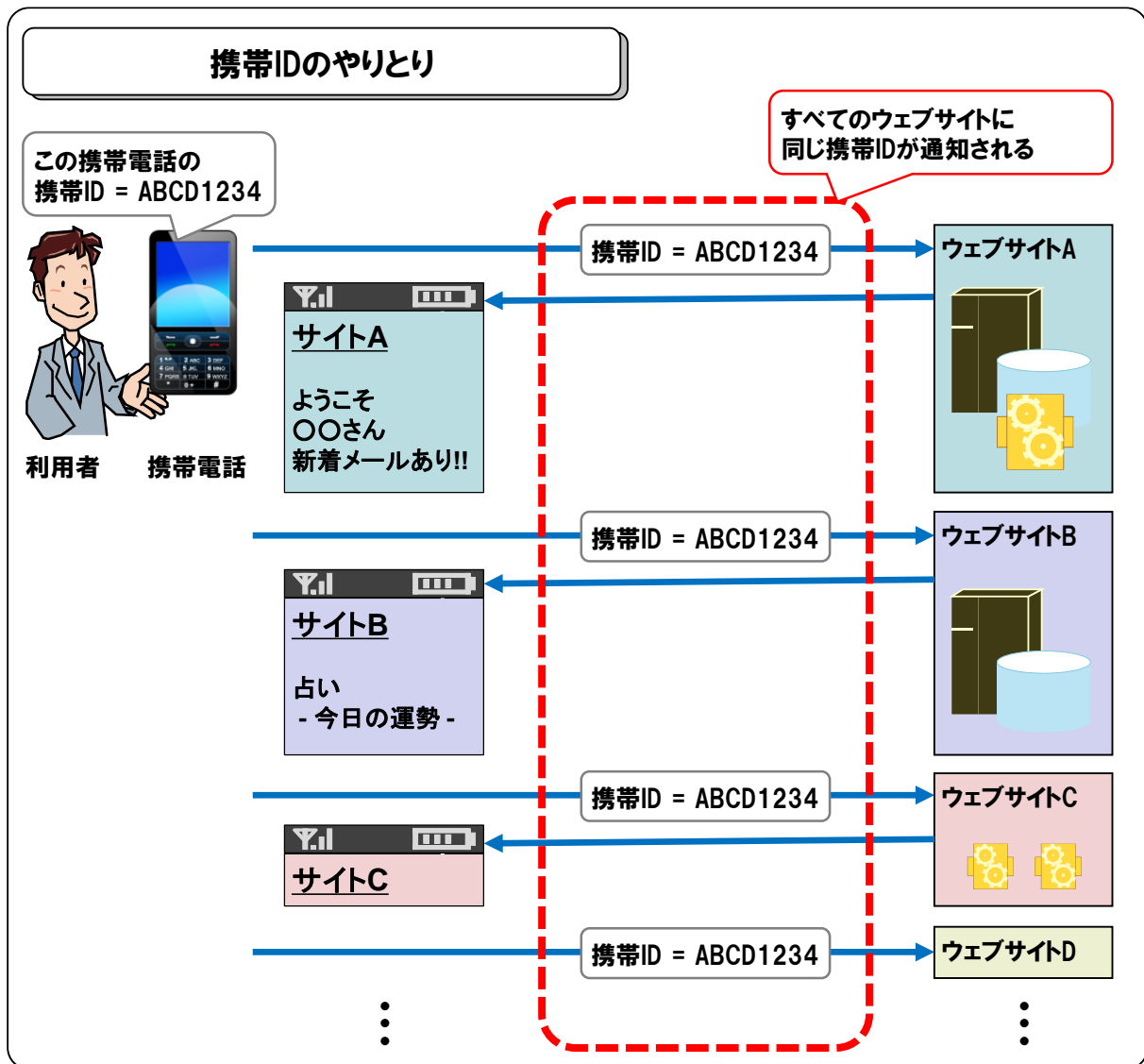
携帯ウェブのブラウザが JavaScript に対応していなかった時代には、携帯ウェブ向けのサイトにおいてクロスサイト・スクリプティング対策が不要と考えられることがありました。しかし、今日では携帯ウェブのブラウザによる JavaScript 対応も進みつつあることから、PC 向けウェブサイトと同様に、クロスサイト・スクリプティング対策が必要です。

2.7.3 携帯 ID の使用に関する注意点

■ 携帯 ID とは

利用者が携帯電話でウェブサイトを開覧する際に、その端末や契約者ごとに割り振られた携帯電話の識別子(以下、「携帯 ID」とする)がウェブサイトへ通知されることがあります。携帯 ID の正式名称はキャリアによって異なり、代表的なものに「i モード ID」、「EZ 番号」、「ユーザ ID」、「FOMA 端末製造番号」、「FOMA カード製造番号」、「端末シリアル番号」などがあります。携帯 ID には、次のような特徴があります。

- (1) すべてのウェブサイトへ同じ携帯 ID が通知される。
- (2) キャリアの公式サイトでなくとも通知される。
- (3) HTTP リクエストヘッダ (User-Agent ヘッダまたは、キャリア独自の拡張ヘッダ) に格納されて、ウェブサイトへ通知される。
- (4) 利用者の設定変更により、通知を停止することができるが、初期設定では通知される。



携帯 ID は、当初、キャリアによっては、いわゆる公式サイトに対してのみ通知されるものでした。しかし、2008 年 3 月以降、すべてのキャリアですべてのウェブサイトにも携帯 ID が通知されるようになったことから、利用が急速にすすみ、携帯 ID に関する脆弱性を抱えたウェブサイトが散見されるようになりました。

■ 携帯 ID による脆弱な認証

ウェブサイトによっては、携帯 ID だけで利用者を認証する設計のものがあります。このような認証方式は、しばしば「かんたんログイン」と呼ばれます。しかし携帯 ID は、すべてのウェブサイトにも送信されるものですので、いわば公開情報です。このため、携帯 ID を照合するだけでは、利用者を認証したことにはなりません。かつて、携帯ウェブは次の 2 つの前提が成り立つと考えられていたことから、携帯 ID を用いて利用者の認証が可能と考えられていました。

- (a) ウェブサイトへのアクセスは、携帯ウェブのブラウザからのみ行われる。または、携帯ウェブのブラウザ以外からのアクセスをウェブサイト側で識別できる。
- (b) 携帯ウェブのブラウザから、利用者による操作で、送信する HTTP リクエストのヘッダを任意に変更することができない。

しかし、近年、このような前提は実際には成り立たなくなってきました。

(a) の前提を満たすために、キャリアが提供している IP アドレスリストを使用し、アクセス元 IP アドレスに基づく制限をする方法が、しばしば用いられます。しかし、このようなリストは、キャリアが正しさを保証していなかったり、安全な取得方法が提供されていなかったり、更新のタイミングを適切に追うことができない等、様々な問題を抱えています。その上、近年では一部のキャリアにおいて、PC を用いてそれらの IP アドレスからのアクセスが可能になっています。

携帯 ID による利用者認証が安全であるためには、ウェブアプリケーションに届く携帯 ID が偽装されないことが必要ですが、上記の通り、(a) の前提が崩れているため、あるキャリアでは端末に割り振られた携帯 ID の偽装を回避できないこと、また、契約者に割り振られた携帯 ID も、一部のキャリアではウェブアプリケーションの実装によっては偽装されてしまうことが知られています⁴²。また、一般にスマートフォン⁴³では簡単に偽装されてしまいます。

このように、携帯 ID を用いて利用者を認証することは簡単ではありません。パスワードや Cookie 等を使用した、PC 向けサイトと同様の認証方式を採用するか、キャリアが提供する安全な認証方式を採用してください。キャリアによって認定された、いわゆる公式サイトでは、キャリアから携帯 ID の安全な使い方に関する情報の開示を受けられることがあります。しかしそれ以外のサイトでは、その情報を得られないため、安全な使い方を把握できず、結果としてウェブサイトの安全性が損なわれる場合があります。認証方式については次項も参照ください。

2.7.4 認証情報に関する注意点

本項では、携帯ウェブ向けのサイトで発生しやすい、認証情報(ウェブサイトが利用者を認証するために使用する情報)に関する問題をとりあげます。

■ 秘密情報ではないものを認証情報として使用

認証情報は、「パスワード」や「暗証番号」など、ウェブサイトと利用者の間だけで共有される秘密情報でなくてはなりません。

たとえば利用者の生年月日など、秘密情報ではないものに基づいて利用者を認証しようとするウェブサイトがありますが、生年月日はその利用者でない人も知っている可能性がある情報ですので、これは認証情報として使用することのできない情報です。このような情報を利用して利用者を安全に認証できま

⁴² 携帯電話向け Web におけるセッション管理の脆弱性

<https://staff.aist.go.jp/takagi.hiromitsu/paper/scis2011-IB2-2-takagi.pdf>

⁴³ 本節におけるスマートフォンとは、端末のブラウザがキャリアのゲートウェイを介さずに直接 HTTP でウェブサイトにつながるものを指します。

せん⁴⁴。

■ 認証強度が足りない場合

認証情報が、第三者による推測や試行によって破られないことがないよう、ウェブサイトは、利用者が十分に複雑な認証情報を使用できるようにする必要があります。

携帯電話の入力インターフェースは PC と異なる方式で、長い文字列の入力には向いていません。このため携帯電話向けウェブサイトにおいては、入力する認証情報を数字のみにするといった設計がなされがちです。しかし、数字のみによる認証は、簡単に破られてしまう場合があります。

たとえば、利用者の認証に使用する情報として数字 4 桁の暗証番号を用いる場合、暗証番号の組み合わせの総数は 10,000 パターンしかありませんから、10,000 パターンの暗証番号を試された場合には高い確率で認証を破られてしまいます。

4 桁の暗証番号による認証は、銀行の ATM や、電話越しでの本人確認で成立しているため、一見すると安全そうです。しかし、そういった本人確認が成立する背景には、試行回数が制限されている前提があり、この前提が損なわれると安全な認証ができなくなります。

ウェブでは多くの場合、試行回数を確実に制限することは困難です。たとえば、試行回数を制限する単純な方法として、あるユーザ ID に対するパスワードの間違いが一定回数を超えた場合には、アカウントをロックするといった対策が考えられますが、このような単純な対策では、パスワードを固定してユーザ ID を変更する方式の試行(リバースブルートフォース)に効果がありません。

ウェブにおける利用者の認証では、認証情報だけが頼りになります。認証情報を数字だけに制限したりせず、英数字を織り交ぜた桁数の多いパスワードを使用できるようにしてください。

■ 利便性との両立

パターン数の多いパスワードは、利用者から見れば入力の手間を要するものです。このためウェブサイト設計の際、利便性を優先してパスワードのパターン数を少なくする方向性の設計に傾くことがあるかもしれません。しかし、利用者の安全性も考慮し、パターン数を確保したまま入力頻度を減らす設計

⁴⁴ 生年月日や電話番号などは、不正アクセス行為の禁止等に関する法律 第二条で規定される識別符号(みだりに第三者に知らせてはならないものとされている符号)に該当しないため、それらのみを認証情報として使用しているウェブサイトは、アクセス制御機能による保護のないサイトとみなされかねない問題もあります。

を検討してください。

入力頻度を減らす方法は幾つかありますが、代表的なものは、Cookie 機能を用いて一定期間有効なセッション ID⁴⁵を発行し、そのセッション ID が有効な間は認証済みとみなす手法です。PC 向けのウェブサイトではしばしば、「次回から自動的にログイン」「ログイン状態を保持する」等の説明の下、利用者の選択でこの機能を使用できる仕組みが提供されています。

セッション ID の有効期間を長くするほど、パスワードの入力頻度を抑えることができます。具体的な有効期間はウェブサイトのサービス内容に応じて個別に検討してください。

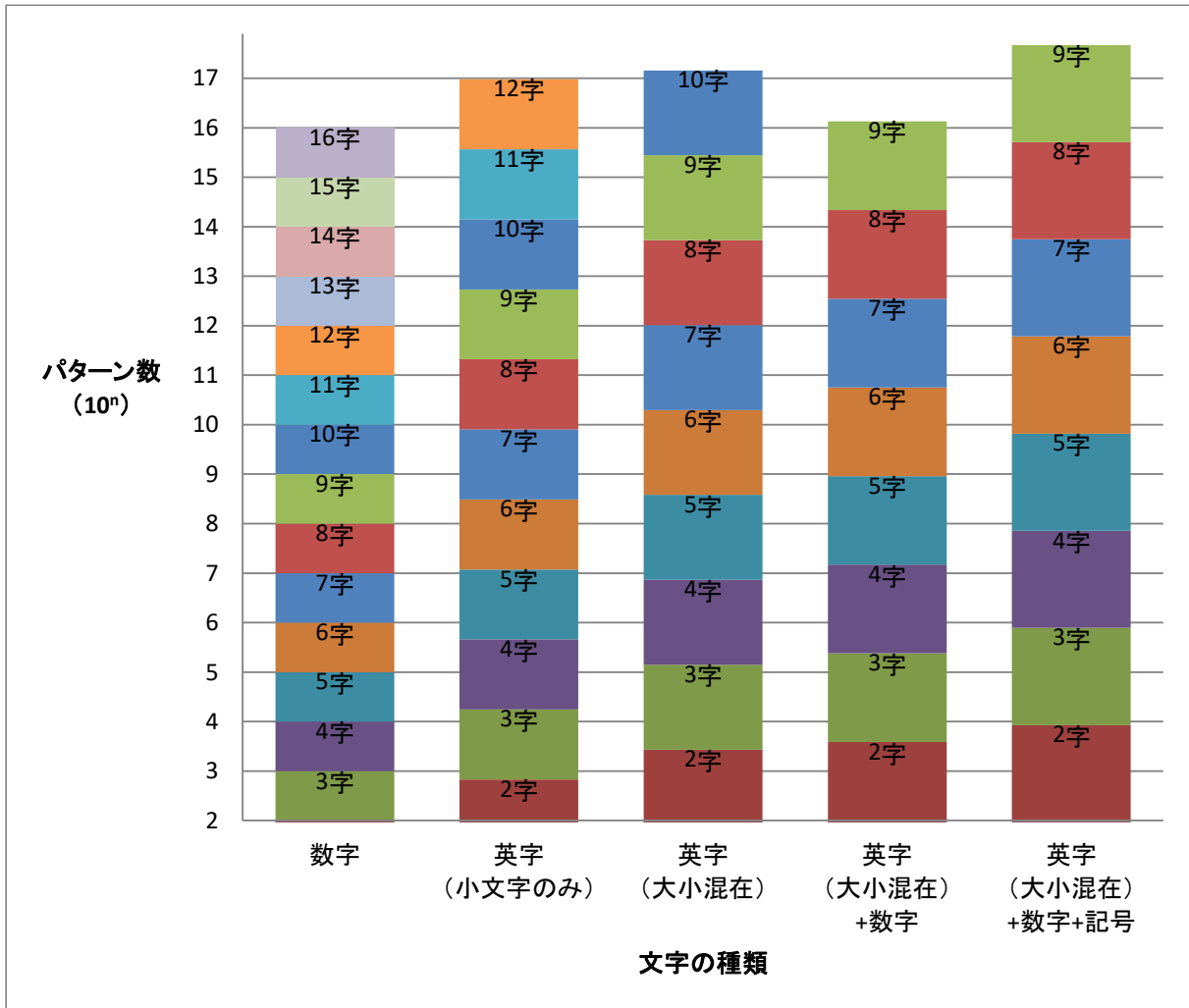
■ パスワードに用いる文字の種類とパターン数の関係

PC 向けのウェブサイトでは、英数字と記号文字を織り交ぜたパスワードを使用するよう、しばしば推奨されます。このようなパスワードには高い強度がありますが、携帯電話で入力することは現実的ではない場合があります。携帯電話においては、数字のみをパスワードにすることが現実的な方法として考えられますが、その場合は桁数を多くして、十分なパターン数を確保する必要があります。代表的なパスワード構成方法ごとのパターン数を図にまとめましたので、これを参考に必要なパターン数を確保してください。

例えば、英字(大小混在)+数字+記号で構成された 8 字のパスワードと同じパターン数を得るためには、数字 16 桁が必要になります。また、数字 4 桁のパスワードでは、英字(大小混在)+数字+

⁴⁵ ここで使用するセッション ID は、第三者に推測されない必要があります。詳しくは 18 ページの根本的解決 4-(i)を参照してください。

記号で構成された2字のパスワードと同程度のパターン数しかないことがわかります。



3. 失敗例

前章までで、ウェブアプリケーションにおける脆弱性対策と、ウェブサイトにおける安全性向上のための取り組みを紹介してきました。本章では、ウェブアプリケーションに脆弱性を作りこんでしまった「失敗例」および、その修正例を紹介します。

3.1 SQL インジェクションの例

SQL インジェクションの脆弱性を考慮できていない例として、ユーザ認証のプログラムを紹介します。

■ PHP と PostgreSQL の組み合わせ

【脆弱な実装】

```
$query = "SELECT * FROM usr WHERE uid = '$uid' AND pass = '$passh' ;
$result = pg_query($conn, $query);
```

PHP

上記はユーザ認証に関するソースコードの一部です。

1 行目右辺の \$uid は、ユーザによって入力されたユーザ ID の値です。また、\$passh は、ユーザによって入力されたパスワードをウェブアプリケーション内でハッシュ処理した値です。1 行目の式では、これらの変数を用い、実行する SQL 文を \$query に代入しています。2 行目右辺の pg_query()⁴⁶ は、PHP に用意されている PostgreSQL 専用の関数で、第 2 引数に指定された \$query を SQL 文として実行します。しかし、このプログラム例では、\$uid に対するエスケープ処理が欠落しています。このため、\$uid に悪意ある SQL 文が形成される値が指定された場合、SQL インジェクション攻撃が成功してしまいます。

【解説】

本例のように、ウェブアプリケーションに外部から渡されるパラメータに対してエスケープ処理を行っていない場合、想定外の SQL 文を実行させられる原因となります。

たとえば、ユーザ ID に「taro' --」という文字列が与えられた場合、ウェブアプリケーションがデータベースに要求する SQL 文は下記のようにになります。

```
SELECT * FROM usr WHERE uid = 'taro' --' AND pass = 'eefd5bc2...'
```

SQL

上記 SQL 文中のシングルクォート「'」は、文字列定数を括る「引用符」の意味を持つ特別な文字です。また、ハイフンの繰り返し「--」は、それ以降の内容をコメントとして無視させる意味をもちます。このため、この文字列が与えられた場合、データベースは「' AND pass = eefd5bc2...」を無視します。この結果、データベースで実行される SQL 文は、下記のようにになります。

⁴⁶ pg_query: <https://www.php.net/manual/ja/function.pg-query.php>

```
SELECT * FROM usr WHERE uid = 'taro' —
```

SQL

これは、仮に「taro」というユーザが存在していた場合、「taro」のパスワードを知らなくてもログインが可能であることを意味します。認証回避だけでなく、\$uid に与える文字列を変えることで、攻撃者は自由にデータベースを操作することができてしまう場合があります。SQL 文を構成する要素に対し、エスケープ処理を施していないことが、本問題の原因です。

なお、pg_query() 関数は、複数のクエリ実行が可能な関数です。この箇所に SQL インジェクションの脆弱性がある場合、もとのクエリとは別に新規のクエリを挿入される等、攻撃による脅威が高まります。下記は、複数の SQL 文の実行例です。

```
// $query に二つの SQL 文を指定
$query = "SELECT item FROM shop WHERE id = 1;
         SELECT item FROM shop WHERE id = 2;";
$result = pg_query($conn, $query);
```

PHP

【修正例 1】

プリペアドステートメントを使う

pg_query() の代わりに、pg_prepare()⁴⁷ および pg_execute()⁴⁸ を利用する

```
$result = pg_prepare($conn, "query", 'SELECT * FROM usr WHERE uid= $1 AND pass=$2');
$result = pg_execute($conn, "query", array($uid, $passh));
```

PHP

pg_prepare() および pg_execute() は、PHP 5.1.0 以降 に用意されている PostgreSQL 用の関数です。PostgreSQL 7.4 以降で利用することができます。

pg_prepare() は、プリペアドステートメント(準備された SQL 文)を作成する関数です。第 3 引数に、実際の値がまだ割り当てられていないパラメータ(プレースホルダ)を含む SQL 文の文字列を指定します。パラメータは「\$1」と「\$2」等の形式で参照されます。

pg_execute() は、pg_prepare() で作成したプリペアドステートメントを実行する関数です。プリペアドステートメントにプレースホルダが存在する場合、pg_execute() は、第 3 引数の要素(\$uid と \$passh)を、自動的に文字列に変換した上でプレースホルダに割り当て(バインド)、完成した SQL 文を実行します。この処理により、利用者は SQL 文を構成する要素に別途エスケープ処理を行う必要がなくなります。

⁴⁷ pg_prepare: <https://www.php.net/manual/ja/function.pg-prepare.php>

⁴⁸ pg_execute: <https://www.php.net/manual/ja/function.pg-execute.php>

【修正例 2】**プレースホルダの仕組みを持つ関数を利用**

`pg_query()` の代わりに、`pg_query_params()`⁴⁹ を利用する

```
$result = pg_query_params($conn, 'SELECT * FROM usr WHERE uid = $1
                                AND pass = $2', array($uid, $passh));
```

PHP

`pg_query_params()` は、PHP 5.1.0 以降⁵⁰に用意されている PostgreSQL 用の関数です。PostgreSQL 7.4 以降で利用することができます。

`pg_query_params()` は、プリペアドステートメントを構成するものではありませんが、プレースホルダの仕組みを持つ関数です。第 2 引数に、プレースホルダ(「\$1」や「\$2」)を含む SQL 文を指定し、第 3 引数に実際の値を指定します。プレースホルダを利用することにより、利用者は SQL 文の要素に対するエスケープ処理を別途行う必要がなくなります。

【修正例 3】**専用のエスケープ関数を利用**

`pg_escape_string()`⁵¹ を利用し、`pg_query()` で実行する SQL 文中の全ての変数要素に対してエスケープ処理を行う

```
$query = "SELECT * FROM usr WHERE uid = '". pg_escape_string($uid). "'
        AND pass = '". pg_escape_string($passh). "'";
$result = pg_query($conn, $query);
```

PHP

`pg_escape_string()` は、PHP 4.2.0 以降に用意されている PostgreSQL 用の関数です。PostgreSQL 7.2 以降で利用することができ、PostgreSQL において特別な意味を持つ文字をエスケープします。

エスケープ処理関数を自作する方法もありますが、PostgreSQL 独自の特別な意味を持つ文字の全てに対応することは難しく、漏れが生じる可能性があるため、お勧めできません。`pg_escape_string()` を用いれば、必要なエスケープ処理を自動的に行ってくれます。

なお、上記コーディングでは、`$passh` に対してもエスケープ処理を行っています。`$passh` は、外部から与えられたパスワードをハッシュ処理した値であるため、この要素が SQL インジェクション攻撃に悪用される可能性は極めて低いと評価できます。しかし、`$passh` のように、内部処理された要素に対しても、あえてエスケープ処理を施すことをお勧めします。これは、エスケープが必要な要素であるかどうかの検討を都度しなくてよいという利点があります。複雑なプログラムにおいては、それぞれの要素に対し、エスケープ処理の要不要判断を行うことは容易ではなく、漏れが生じる原因となります。SQL 文を構成する全ての変数要素に対し、一貫してエスケープ処理を行うことをお勧めします。

⁴⁹ `pg_query_params`: <https://www.php.net/manual/ja/function.pg-query-params.php>

⁵⁰ PHP 5.3 は、2014 年 8 月 14 日でサポートが終了しています。サポート中のバージョンの利用を推奨します。
<https://www.php.net/eol.php>

⁵¹ `pg_escape_string`: <https://www.php.net/manual/ja/function.pg-escape-string.php>

■ PHP と MySQL の組み合わせ

【脆弱な実装】

```
$query = "SELECT * FROM usr WHERE uid = '$uid' AND pass = '$passh'";
$result = mysql_query($query);
```

PHP

上記はユーザ認証に関するソースコードの一部です。

本例も、前述の PHP と PostgreSQL の組み合わせと同様、\$uid に対するエスケープ処理が欠落しています。このため、\$uid に悪意ある SQL 文が形成される値が指定された場合、SQL インジェクション攻撃が成功してしまいます。

【修正例 1】

プリペアドステートメントを使う

mysql_query() の代わりに、mysqli(MySQL 拡張サポート)⁵² の mysqli_prepare()⁵³、mysqli_stmt_bind_param()⁵⁴、mysqli_stmt_execute()⁵⁵ 関数等を利用する

```
// プリペアドステートメントの作成
$stmt = mysqli_prepare($conn, "SELECT * FROM usr WHERE uid= ? AND pass = ?");
// プレースホルダに $uid, $passh をバインド
mysqli_stmt_bind_param($stmt, "ss", $uid, $passh);
// SQL 文の実行
mysqli_stmt_execute($stmt);
```

PHP

mysqli_prepare()、mysqli_stmt_bind_param()、mysqli_stmt_execute() は、PHP の mysqli モジュールに用意されている MySQL 用の関数です。mysqli は、MySQL 4.1.3 以上の環境で利用することができます。

mysqli_prepare() は、プリペアドステートメント(準備された SQL 文のひな型)を作成する関数です。第 2 引数の値が、プリペアドステートメントの内容です。この文字列のうち、「?」は、「プレースホルダ」と呼ばれる、実際の値がまだ割り当てられていない要素です。

mysqli_stmt_bind_param() は、mysqli_prepare() で作成されたプリペアドステートメントのプレースホルダに、対応する値(バインド値)を割り当てる関数です。第 3 引数以降の要素(「\$uid」と「\$passh」)が、バインド値に相当します。第 2 引数の「ss」は、バインド値の型を指定するものです。「\$uid」と「\$passh」はともに文字列型であるため、string の頭文字である「s」を 2 要素分並べます。

mysqli_stmt_execute() は、完成したプリペアドステートメントを実行する関数です。これらの関数を利用することにより、利用者は SQL 文の要素に対するエスケープ処理を別途行う必要がなくなります。

また、mysql 関数は PHP 5.5.0 にて非推奨となり、PHP 7.0.0 で削除されたため、代替の関数に変更すべきです。

⁵² MySQL 改良版拡張サポート (mysqli) <https://www.php.net/mysqli/>

⁵³ mysqli_prepare: <https://www.php.net/manual/ja/mysqli.prepare.php>

⁵⁴ mysqli_stmt_bind_param: <https://www.php.net/manual/ja/mysqli-stmt.bind-param.php>

⁵⁵ mysqli_stmt_execute: <https://www.php.net/manual/ja/mysqli-stmt.execute.php>

【修正例 2】**エスケープ関数を利用**

`mysql_query()` の代わりに、`mysqli` (MySQL 拡張サポート) の `mysqli_query()`⁵⁶ を利用し、`mysqli_real_escape_string()`⁵⁷ により、`mysqli_query()` で実行する SQL 文中の全ての変数要素に対し、エスケープ処理を行う

```
$query = "SELECT * FROM usr WHERE uid = '".
    mysqli_real_escape_string($conn, $uid)."' AND pass = '".
    mysqli_real_escape_string($conn, $passh)."'";
$result = mysqli_query($conn, $query);
```

PHP

`mysqli_real_escape_string()` は、PHP 5.0.0 以降に用意されている MySQL 用の関数です。この関数は、MySQL において特別な意味を持つ文字をエスケープします。

エスケープ関数を自作することは、漏れが生じる可能性があるため、お勧めできません。また、対策漏れ防止の観点より、`$passh` のように、内部処理された要素に対しても、一貫してエスケープ処理を施すことをお勧めします。

■ Perl (DBI を利用)**【脆弱な実装】**

```
$query = "SELECT * FROM usr WHERE uid = '$uid' AND pass = '$passh'";
$sth = $dbh->prepare($query);
$sth->execute();
```

Perl

上記はユーザ認証に関するソースコードの一部です。本例では、Perl において広く利用されている DBI⁵⁸と呼ばれる、データベースへアクセスするためのモジュールを使用しています。

データベースへのアクセスは、DBI モジュールのデータベースハンドルメソッド(`prepare()` 等)やステートメントハンドルメソッド(`execute()` 等)を使用します。しかし、本例では `$uid` に対するエスケープ処理が欠落しています。このため、`$uid` に悪意ある SQL 文が形成される値が指定された場合、SQL インジェクション攻撃が成功してしまいます。

【解説】

この例は、Perl による実装で DBI を用いている場合によく見かける、危険なコーディング例です。

DBI モジュールの `prepare()` メソッドは、プリペアドステートメントを作成するメソッドで、プレースホルダを利用することができます。また、`execute()` メソッドは、`prepare()` メソッドで作成されたプリペアドステートメントを実行するメソッドで、プリペアドステートメントにプレースホルダが存在する場合、その場所にバインド値を割り当てます。

しかし、本例では、実行する SQL 文に変数要素を含むにも関わらず、プレースホルダを使用しません。また SQL 文を構成する要素に対してエスケープ処理を行っていません。このため、SQL イン

⁵⁶ `mysqli_query`: <https://www.php.net/manual/ja/mysqli.query.php>

⁵⁷ `mysqli_real_escape_string`: <https://www.php.net/manual/ja/function.mysql-real-escape-string.php>

⁵⁸ DBI: <https://dbi.perl.org/about/>

ジェクション攻撃に脆弱となります。

【修正例 1】

プリペアドステートメントでプレースホルダを使う

```
$sth = $dbh->prepare("SELECT * FROM usr WHERE uid = ? AND pass = ?");
$sth->execute($uid, $passh);
```

Perl

DBI モジュールの `prepare()` メソッドに SQL 文を指定する際、プレースホルダを利用し、変数に相当する箇所を「?」とします。また、`execute()` メソッドには、プレースホルダに割り当てる値(バインド値)を指定します。

【修正例 2】

エスケープ関数を利用

エスケープ対象の要素に対し、DBI モジュールの `quote()` メソッドを使用します。

```
$sth = $dbh->prepare("SELECT * FROM usr
                    WHERE uid = ".$dbh->quote($uid)." AND
                    pass = ".$dbh->quote($passh));
$sth->execute();
```

Perl

`quote()` メソッドは、引数に指定された文字列に対して、特別な意味を持つ文字をエスケープ処理したうえで、それ全体をクオートで囲んだ文字列を返します。

SQL 文においてエスケープ処理を行う場合には、通常、特別な意味を持つ文字がデータベースエンジン毎に異なることに対応しなければなりません。DBI には、DBD (DataBase Drivers)と呼ばれる、各種データベースエンジンに対応したドライバが用意されており、DBI の `quote()` メソッドは、DBD にその処理を委譲しているため、データベースエンジンによる違いを意識せずに利用することができます。

3.2 OS コマンド・インジェクションの例

OS コマンド・インジェクションの脆弱性を考慮できていない例として、メール送信のプログラムを紹介します。

■ Perl から sendmail コマンドの呼び出し

【脆弱な実装】

```
$from =~ s/"|'|<|>|¥| | //ig;
open(MAIL, "|/usr/sbin/sendmail -t -i -f $from");
```

Perl

これは、ウェブのフォームに入力されたメールアドレスを差出人としてメールを送信するプログラムの一部です。

\$from に、入力された差出人アドレスが格納されています。1 行目は、シェルのコマンドライン上で特別な意味を持つ文字である「"」、「'」、「<」、「>」、「|」、「¥」を \$from から削除しようとしています。2 行目は、OS の sendmail コマンドを呼び出して、メールを送信する処理を開始し、差出人アドレスとして \$from の値をコマンドライン引数に渡しています。

この実装は、1 行目の処置を施してもなお、OS コマンド・インジェクション攻撃に対して脆弱です。

【解説】

この実装で、\$from の値が「someone@example.jp」であるならば、次のコマンドが実行され、これは正常に処理されます。

```
/usr/sbin/sendmail -t -i -f someone@example.jp
```

sh

しかし、攻撃を意図した入力により、\$from の値が「`touch[0x09]/tmp/foo`」(ここで「[0x09]」は水平タブを表す)となった場合、次のコマンドが実行され、OS コマンド・インジェクションの脆弱性を突いた攻撃が成立してしまいます。

```
/usr/sbin/sendmail -t -i -f `touch[0x09]/tmp/foo`
```

sh

バッククォート「`」は、囲まれた部分を実行してその出力をコマンドラインに反映するという、UNIX におけるシェルの機能です。例題コードでは、ダブルクォートやシングルクォートは削除していましたが、バッククォートは削除していませんでした。そのため、攻撃者が指定したコマンドの実行を許してしまっています。

また、例題コードでは、空白文字を削除していましたが、攻撃者は任意のコマンドを実行することができても、コマンド引数を自由に指定することはできないと思われるところですが、上記のように、水平タブ「[0x09]」を使うことで、任意のコマンド引数を指定することができてしまいます。ここで、水平タブは空白文字と同様、区切り文字として意味を持ちます。

どの文字がシェル上で特別な意味を持つかはシェルの種類によって異なります。思いつきで適当な文字を削除する方法では、不完全な対策となる可能性が高いため、注意が必要です。

【修正例 1】**ライブラリを使用する方法**

コマンドの呼び出しをやめることで、OS コマンド・インジェクションの脆弱性に対する根本的解決となります。コマンドの呼び出しで実現していた機能を、既存のライブラリを用いて実現できないか検討してください。

```
use Mail::Sendmail;
%mail = (From => $from, ...);
sendmail(%mail);
```

Perl

例題コードでは、メールを送信することが目的ですから、メールを送信するライブラリ「Mail::Sendmail」を利用すれば、機能を維持したまま OS コマンド・インジェクションの脆弱性を解消できます。

【修正例 2】**コマンドライン中に値を埋め込まない方法**

ライブラリの利用が難しく、コマンドを使わざるを得ない場合でも、コマンドの呼び出し方を変更することで、OS コマンド・インジェクションの脆弱性を解消できる場合があります。

```
$from =~ s/¥r|¥n//ig;
open(MAIL, '|/usr/sbin/sendmail -t -i');
...
print MAIL "From: $from¥n";
```

Perl

例題コードの場合、メールの差出人を指定する部分が問題となっていました。sendmail コマンドでは、差出人は必ずしも引数で指定する必要はありません。上記のようにコマンドの標準入力に与えるメールデータのヘッダに差出人を指定することができます。この方法ならば、コマンドライン中に値を埋め込むことを避けられますので、OS コマンド・インジェクションの脆弱性を解消できます。

ただし、この修正例のように、メールヘッダを出力する際には、メールヘッダ・インジェクションの脆弱性に注意が必要で、出力する \$from に改行コードが含まれないようにしなければなりません。3.8 の修正例 2 もあわせて参照ください。

【修正例 3】**シェルを経由せずにコマンドを呼び出す方法**

ライブラリの利用が難しく、コマンドを使わざるを得ない場合でも、シェルを経由せずにコマンドを呼び出すことで、OS コマンド・インジェクションの脆弱性を解消できる場合があります。

```
open(MAIL, '|-') || exec '/usr/sbin/sendmail', '-t', '-i', '-f', '$from';
```

Perl

Perl では、上記コードによりシェルを経由せずに直接コマンドを起動します。このコードは、例題コードの 2 行目と同じ機能を実現します。このため、\$from にシェル上で特別な意味を持つ文字が含まれていても、シェルの機能が実行されないため、OS コマンド・インジェクションの脆弱性を解消できます。

3.3 パス名パラメータの未チェックの例

パス名パラメータに関する脆弱性を考慮できていない例として、ファイル内容を画面表示するプログラムを紹介します。

■ PHP によるファイル内容の画面表示

【脆弱な実装】

```
$file_name = $_GET['file_name'];
if(!file_exists($file_name)) {
    $file_name = 'nofile.png';
}
$fp = fopen($file_name, 'rb');
fpassthru($fp);
```

PHP

これは、指定された名前のファイルの内容を画面に表示するプログラムの一部です。1 行目の `$file_name` には、URL 中の `file_name` パラメータで指定されたファイル名が代入されます。そのファイルが存在する場合、5 行目の `fopen()` で開き、内容を 6 行目の `fpassthru()` で出力します。指定されたファイルが存在しない場合、`nofile.png` を出力します。ここでは、指定されるファイルはサーバの公開ディレクトリ上にあるファイルのみと想定しています。

この実装は、URL 中で指定されるファイル名に、絶対パス名や、「../」を含むパス名が与えられる可能性があることを考慮していないため、ディレクトリ・トラバーサル攻撃に対して脆弱です。

【解説】

この実装では、URL 中の `file_name` パラメータで「/etc/passwd」を指定された場合、/etc/passwd の内容を画面に表示してしまいます。

また、下記のように、ディレクトリをプログラム中で指定することにより、URL で絶対パス名を指定されることを防止したとしても、URL で「../../../../etc/passwd」のように、上位ディレクトリを辿る相対パス名を指定されると、/etc/passwd の内容を表示してしまいます。

```
$file_name = $_GET['file_name'];
$dir = '/home/www/image/'; //ディレクトリを指定
$file_path = $dir . $file_name;
if(!file_exists($file_path)) {
    $file_path = $dir . 'nofile.png';
}
$fp = fopen($file_path, 'rb');
fpassthru($fp);
```

PHP

【修正例】

パス名からファイル名だけを取り出して使用する

OS や言語に用意されている機能を用いて、パス名からファイル名部分だけを取り出して使うようにすることで、パス名パラメータに関する脆弱性に対する根本的解決となります。

```
$dir = '/home/www/image/';  
...  
$file_name = $_GET['file_name'];  
...  
if(!file_exists($dir . basename($file_name))) {  
    $file_name = 'nofile.png';  
}  
$fp = fopen($dir . basename($file_name), 'rb');  
fpassthru($fp);
```

PHP

`basename()` は引数のパス名からファイル名部分(ディレクトリ部分を含まない)を取り出す関数です。`basename()` を使用することで、絶対パス名や「../」を含むパス名が指定された場合でも、ファイル名のみを取り出して使用することができます。これにより、パス名パラメータに関する脆弱性を解消できます。

3.4 不適切なセッション管理の例

不適切なセッション管理の実装例として、セッション ID を生成するプログラムを紹介します。

■ Perl によるセッション ID の生成

【脆弱な実装】

```
sub getNewSessionId {
    my $sessid = getLastSessionId ('/tmp/.sessionid');
    $sessid++;
    updateLastSessionId ('/tmp/.sessionid', $sessid);
    return $sessid;
}
```

Perl

これはセッション ID を発行するプログラムの一部です。このプログラムでは `getNewSessionId` 関数を呼び出してセッション ID を生成しています。`getNewSessionId` 関数では、ファイル `/tmp/.sessionid` に保存している数値を 1 ずつ増やしながらセッション ID を返します。

この実装では、セッション ID が推測可能となる脆弱性があります。

【解説】

この実装では、セッション ID を数値で表しており、1 から始めて、2、3、4 と連番で発行しています。このプログラムでは、最後に発行したセッション ID をファイル `/tmp/.sessionid` で管理しています。攻撃者がこのサイトを利用すると、攻撃者にもセッション ID が発行されます。たとえば、攻撃者が取得したセッション ID が「3022」であったなら、そのタイミングで「3021」のセッション ID が有効である可能性があります。攻撃者は、セッション ID として「3021」を送信してサイトにアクセスすることで、セッション ID 「3021」が割り当てられている他の利用者のセッションを乗っ取ることができてしまいます。

こうしたセッション・ハイジャック攻撃を防止するために、セッション ID は、暗号論的擬似乱数生成器を用いて生成するべきです。

【よくある失敗例 1】

第三者が推測可能な値に基づいたセッション ID の生成

```
sub getNewSessionId {
    my $sessid = time() . '_' . $$;
    ...
    return $sessid;
}
```

Perl

このプログラムでは、UNIX 時間⁵⁹とプロセス ID の組み合わせをセッション ID としています。ここでは、アクセスごとに新しいプロセスが生成される CGI 実行方式を想定しています。

このプログラムでセッション ID を生成する `getNewSessionId` 関数を呼び出すと、UNIX 時間(`time` 関数)、アンダースコア「`_`」、プロセス ID (変数 `$$`)を連結して、その連結した文字列をセッション ID として

⁵⁹ 1970 年 1 月 1 日 0 時 0 分 0 秒からの経過秒数を指す。UNIX 時刻やエポック秒と呼ばれることもある。

返します。この関数が返すセッション ID は、例えば「1295247752_27554」等です。

このプログラムは、セッション ID の推測によるセッション・ハイジャック攻撃に対して、脆弱です。

このプログラムで、攻撃者がセッションを確立した時刻から 1 分後までに、10 個の他のセッションが確立されたと仮定します。攻撃者は、まず、自分用に発行されたセッション ID から、自分が接続しているウェブアプリケーションのセッションのプロセス ID が分かります。一般的に新規のプロセスにはプロセス ID が連番で割り当てられることが多いです。攻撃者のプロセス ID が「27554」だった場合、他のセッションのプロセス ID は「27555」、「27556」…「27564」だと推測できます。次に、攻撃者がセッションを確立した UNIX 時間が「1295247752」の場合、この UNIX 時間以降 1 分間の UNIX 時間は「1295247753」から「1295247812」の範囲の値となります。推測したプロセス ID とこれらの UNIX 時間を組み合わせると、600 通りのセッション ID が得られます。推測したセッション ID 600 通りを試行することで、セッション・ハイジャック攻撃が成功する可能性があります。

【よくある失敗例 2】

第三者が知り得る値に基づいたセッション ID の生成

```
use Digest::SHA qw(sha256_hex);
...
sub getNewSessionId {
    my $sessid = '';
    $sessid = $sessid . $ENV{'REMOTE_ADDR'};
    $sessid = $sessid . $ENV{'REMOTE_PORT'};
    $sessid = $sessid . time();
    $sessid = Digest::SHA::sha256_hex($sessid);
    ...
    return $sessid;
}
```

Perl

このプログラムでは、利用者の接続元 IP アドレス、ポート番号、UNIX 時間の組み合わせからハッシュ値を算出し、それをセッション ID としています。

このプログラムで、セッション ID を生成する `getNewSessionId` 関数を呼び出すと、`getNewSessionId` 関数はまず利用者の接続元 IP アドレス (`$ENV{'REMOTE_ADDR'}`)、接続元ポート番号 (`$ENV{'REMOTE_PORT'}`)、UNIX 時間 (`time` 関数) を連結して文字列を作成します。そして、`getNewSessionId` 関数はこの文字列の SHA256 ハッシュを算出し、そのハッシュ値をセッション ID として返します。この関数が返すセッション ID は、例えば「093a2031a79cb4904b1466ee7ad5faaa3afe7b787db66712f407326b213cc2a4」等です。

このプログラムではセッション ID にハッシュ値を使っているため、一見、安全そうに見えるかもしれませんが、しかし、セッション ID の生成方法が第三者に知られた場合⁶⁰には、第三者がセッション ID を推測できる余地があります。

震のウェブサイトへの誘導等によって、攻撃者は利用者の接続元 IP アドレスを入手できます⁶¹。一方、ポート番号は攻撃者が知り得ない情報です。しかし、接続元ポート番号の範囲は 1024 から 65535 であり、利用者のネットワーク環境によってはこの範囲が 2 万通り程度に限定される場合があります。

⁶⁰ このプログラムがオープンソースで開発されている場合、またはソースコードが漏えいしてしまった場合等を想定できます。

⁶¹ ウェブサイトに到達するまでのネットワーク経路によっては、特定の IP アドレスとならない場合があります。

3.4 失敗例(不適切なセッション管理)

接続元ポート番号が 2 万通りに限定されるネットワーク環境において、このプログラムのウェブアプリケーションに利用者が接続してセッションを確立し、その 10 秒前後以内に攻撃者が利用者の接続元 IP アドレスを知ったとすると、接続元ポート番号の取り得る値 2 万通りと、UNIX 時間の 10 通りの組み合わせから、利用者のセッション ID が取り得る値は 20 万通りです。これらのセッション ID を試行することで、セッション・ハイジャック攻撃が成功する可能性があります。

3.5 クロスサイト・スクリプティングの例

クロスサイト・スクリプティングの脆弱性を考慮できていない例として、いくつかのプログラムを紹介しましょう。

クロスサイト・スクリプティングの脆弱性は原理上、根絶が困難な脆弱性です。しかし、そもそも対策を行っていない場合や、誤った対策を実施しているケースも少なくありません。ここでは、失敗例を3つに分けて紹介します。

1. 未対策
2. 対策漏れ
3. 誤った対策

3.5.1 未対策

■ エスケープ処理の未実施

【脆弱な実装】



```
use CGI qw/:standard/;
$keyword = param('keyword');
...
print ... <input name="keyword" type="text" value="$keyword">
      ... 「$keyword」の検索結果...
```

Perl

上記ソースコードは、検索結果の表示処理の一部です。

検索フォームに入力された文字列「IPA」はウェブアプリケーションに送信され、\$keywordに格納されます。このウェブアプリケーションは、検索結果をウェブページとして出力する際、この \$keywordを、フォーム内や見出し等、複数の場所に埋め込んでいます。しかし、この \$keywordに対して、出力前にエスケープ処理を行っていません。このエスケープ処理の未実施が、スクリプトを埋め込まれる原因となります。

【解説】

ウェブアプリケーションが文字列を出力する際には、それをテキストとして出力するのか、HTML タグ

として出力するののかによって、行うべき処理が異なります。この例の場合、\$keyword は検索キーワードであり、テキストとして出力すべき要素です。したがって、\$keyword に含まれる「&」、「<」、「>」、「"」、「'」⁶²等に対して、エスケープ処理を行う必要があります。

この処理を怠ると、\$keyword にこれらの文字を含む文字列が指定されることにより、開発者の意図に反して画面が崩れる不具合が生じます。この不具合を悪用した攻撃手法が、クロスサイト・スクリプティング攻撃です。

【修正例 1】

エスケープ用の関数を利用

CGI モジュールの `escapeHTML()` を利用する

```
use CGI qw/:standard/;
$keyword = param('keyword');
...
print "<input ... value=¥".escapeHTML($keyword)."¥"...";
print "「".escapeHTML($keyword)."」の検索結果...";
```

Perl

`escapeHTML()` は、Perl の拡張モジュール CGI に用意されている関数の一つです。CGI モジュールは Perl5 に標準で組み込まれています。

`escapeHTML()` は、引数に指定された文字列に含まれる、HTML において特別な意味を持つ文字に対してエスケープ処理を行い、その結果を返します。下記は、`escapeHTML()` におけるエスケープ対象文字と、その処理結果です⁶³。

対象文字	処理結果
&	&
<	<
>	>
"	"
'	'

⁶² タグ内の引用符は一般に「"」(ダブルクォート)が使用されますが、「'」(シングルクォート)を引用符として使用するケースも少なくないため、併記しています。

⁶³ CGI モジュールでは、文字コードに応じて細かくエスケープ対象の文字を決定しています。たとえば、文字コードが ISO-8859-1 や WINDOWS-1252 の場合、0x8B(Single Left-Pointing Angle Quotation Mark)や 0x9b(Single Right-Pointing Angle Quotation Mark)等もエスケープ対象になります。

【修正例 2】

独自に作成したエスケープ処理関数を使用する

```
print "<input ... value=¥"". &myEscapeHTML($keyword). "¥"...";
print "「". &myEscapeHTML($keyword). "」の検索結果...";
...

# 独自に作成したエスケープ処理関数 myEscapeHTML
sub myEscapeHTML ($) {
    my $str = $_[0];
    $str =~ s/&/&amp;/g;
    $str =~ s/</&lt;/g;
    $str =~ s/>/&gt;/g;
    $str =~ s/"/&quot;/g;
    $str =~ s/'/&#39;/g;
}
```

Perl

■ 文字コードの未指定

【脆弱な実装】

ウェブアプリケーションの応答結果

```
HTTP/1.1 200 OK
...
Content-Type: text/html
```

① HTTP レスポンスヘッダに文字コードの指定がない

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html">
```

② HTML の META 宣言にも文字コードの指定がない

HTTP レスポンス

上記は、あるウェブアプリケーションの応答結果の一部です。

「Content-Type」フィールドの値は、送信されるデータの種類(メディアタイプ)をウェブブラウザに判定させるために利用する情報です。しかし、上記には、文字コード(charset)を判別するための情報が指定されていません。この場合、ウェブブラウザは、独自の実装に基づく文字コード判定(たとえば、受信したデータの内容から文字コードを推測する)を行います。この挙動がクロスサイト・スクリプティング攻撃に悪用される場合があります。

【解説】

本例は、ウェブブラウザにおける独自の文字コード判定機能を悪用したクロスサイト・スクリプティング攻撃の対策が未実施である例です。この解決には HTTP レスポンスヘッダの「Content-Type」フィールドに文字コードを指定する必要があります。

詳細は 1.5.3「全てのウェブアプリケーションに共通の対策」5-(viii)の内容を参照してください。

【修正例】

HTTP レスポンスヘッダの「Content-Type」フィールドに文字コードを指定

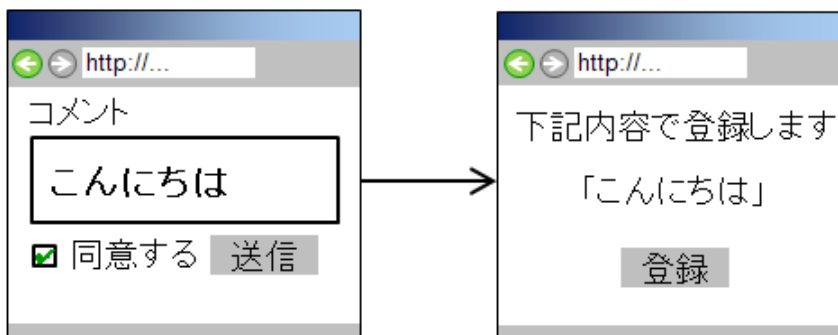
```
HTTP/1.1 200 OK
...
Content-Type: text/html; charset=UTF-8
```

HTTP レスポンス

3.5.2 対策漏れ

■ テキスト形式で入力される値のみに入力段階でエスケープ処理

【脆弱な実装】



本例は、ウェブブラウザにおける独自の文字コード判定機能を悪用したクロスサイト・スクリプティング攻撃の対策が未実施である例です。この解決には HTTP レスポンスヘッダの「Content-Type」フィールドに文字コードを指定する必要があります。

投稿フォーム

```
<textarea name="comment" ...
<input name="agree" type="checkbox" value="yes">...
<input name="uid" type="hidden" value="12345678">...
```

HTML

確認画面

```
$comment = escapeHTML(param('comment'));
$agree   = param('agree');
$uid     = param('uid');
...
print "下記内容で登録します<BR>「". $comment. "」...
print "<input ... hidden ... =¥"". $uid ...
```

Perl

上記は、投稿フォームの HTML ソースと、そのフォームから送信された情報のいくつかを確認画面として出力するウェブアプリケーションのソースの一部です。

投稿フォームには、投稿者が書き込みできるコメント欄と、チェックボックス、非表示情報のユーザ ID が設置されています。投稿フォームから送信されたこれら 3 つの値は、ウェブアプリケーションに渡され、コメント(\$comment)とユーザ ID(\$uid)の 2 つが、確認画面に出力されています。

このうち、コメントに対しては、入力値を受け取る段階でエスケープ処理を行っていますが、ユーザー ID に対してはエスケープ処理を行っていません。これは、エスケープ処理の対象を正しく認識していないために生じる「対策漏れ」の一例です。

【解説】

エスケープ処理対象について、ありがちな誤った認識として、「テキストで入力可能な要素」のみを対象としていることが挙げられます。

攻撃は、フォームのコメント欄のように自由に入力できる項目のみを悪用するわけではありません。テキストで入力可能な要素のみに着目すると、その他の要素を見逃すこととなります。また、攻撃を入力段階で防御しようとする意識が先行して、入力値を受け取った段階でエスケープ処理を行うことも、対策漏れにつながります。クロスサイト・スクリプティングの脆弱性への対策におけるエスケープ対象は「出力要素」であるのに、入力段階でエスケープ処理を行うと、出力前の演算の結果が HTML タグやスクリプトを形成するケースに対応できませんし、必要な部分にエスケープ処理が施されているかどうかをソースコードから確認する作業のコストが増大するデメリットも生じます。

【修正例】

「出力」に注目してエスケープ処理

```

$comment = param('comment');
$agree   = param('agree');
$uid     = param('uid');
...
print escapeHTML($comment);...
print "<input ... hidden ... =¥"".escapeHTML($uid)."..."

```

入力要素には注目しない

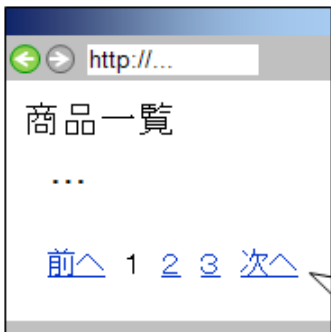
出力する全要素にエスケープ処理

Perl

入力要素に注目せず、出力要素に注目してエスケープ処理を実装してください。

【よくある失敗例 1】

リンクの URL を構成する要素へのエスケープ処理漏れ



```

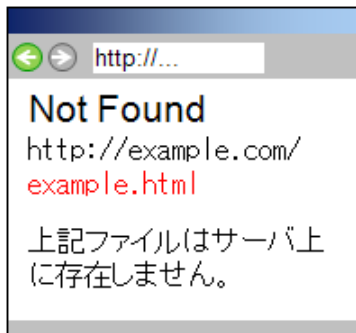
<a href="http://...list.cgi?
cid=1&page=2&pmax=1000&ls=all
...>次へ

```

上図では、「次へ」等のリンクの URL を構成するパラメータとして「cid」、「page」、「pmax」、「ls」等が使用されています。このような、タグ内に出力する要素についてもエスケープ処理を行う必要がありますが、これを見落としてしまう失敗例が少なくありません。

【よくある失敗例 2】

「404 Not Found」ページに表示する URL へのエスケープ処理漏れ



上図は、HTTP ステータスコード 404 用のページとして、ユーザからリクエストされた URL 情報を出力しています。本来、この URL 情報に対してもエスケープ処理を行う必要がありますが、これを見落としてしまうケースが少なからずあります。

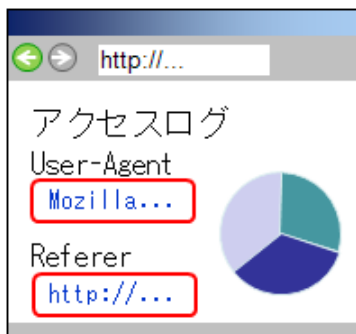
たとえば、下記のような罠のリンクに誘導されることで、クロスサイト・スクリプティング攻撃が成功してしまいます。

```
http://example.com/<script>...</script>
```

URL

【よくある失敗例 3】

アクセスログの出力対象へのエスケープ処理漏れ



本例は、ウェブサーバのアクセスログを基に、統計情報等をウェブページに出力するウェブアプリケーションです。たとえば、リクエストしたページや、User-Agent、Referer 情報等が出力されます。このようなサーバ内のデータを参照して出力する際に、エスケープ処理を見落としてしまうケースが少なからずあります。

たとえば、悪意のある人は、User-Agent や Referer 等の内容にスクリプト文字列を含ませてリクエストをし、アクセスログに記録させます。

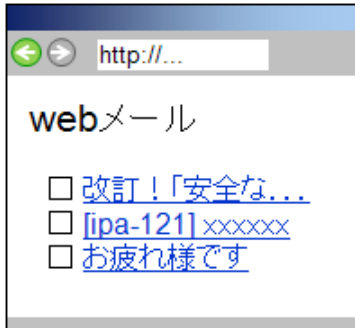
```
GET /example.html / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0...<script>...
Referer: http://example.net/<script>...
```

HTTP リクエスト

このため、エスケープ処理に漏れがある場合、アクセスログのページを閲覧する利用者は、恒久的にスクリプトを埋め込まれたウェブページを閲覧することになります。

【よくある失敗例 4】

ウェブメールの出力対処へのエスケープ処理漏れ



本例は、メールの情報をウェブページに出力するウェブアプリケーションです。たとえば、送信元や件名、メールの内容等が出力されます。このような、サーバ内のデータを参照して出力する際に、エスケープ処理を見落としてしまうケースが少なからずあります。

たとえば、悪意のある人は、下記のようなメールを送信します。

```
宛先: jiro@example.com
From: taro@example.com
件名: 改訂!「安全な... <script>...
内容: IPA です... <script>...
```

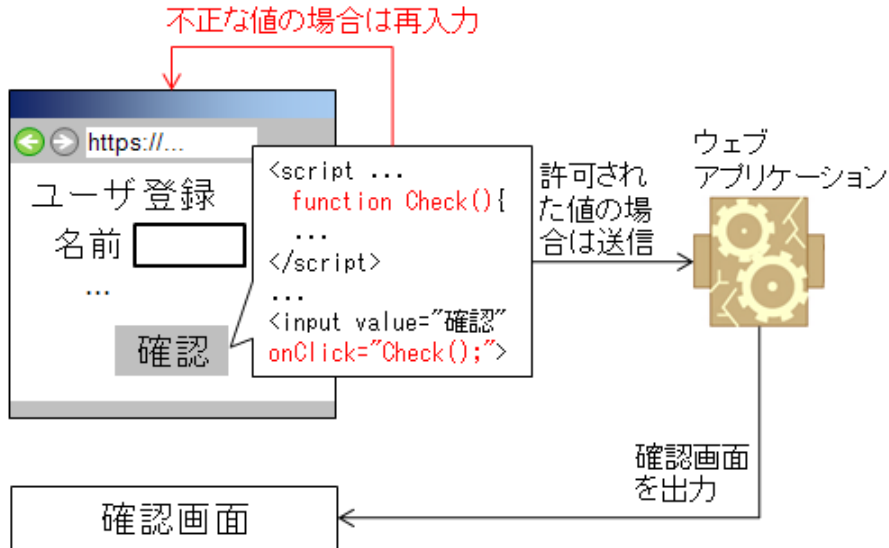
メール

このため、エスケープ処理に漏れがある場合、ウェブメールのページを閲覧する利用者は、恒久的にスクリプトを埋め込まれたウェブページを閲覧することになります。

3.5.3 誤った対策

■ 入力フォームに入力制限を実装

【脆弱な実装】



本例では、入力フォームのウェブページに、JavaScriptによるチェック機構を実装しています。このチェック機構を介すことにより、確認画面を出力するウェブアプリケーションには許可された入力値のみが渡されます。このチェック機構により、確認画面には不正な文字は出力されないと考えがちですが、このチェック機構は、クロスサイト・スクリプティングの脆弱性への対策としては有効に機能しません。

【解説】

対策を実施する箇所が誤っています。入力側(クライアント側)でのチェック機構は、利用者の入力ミスを軽減する目的においては有効に機能しますが、クロスサイト・スクリプティングの脆弱性への対策としては有効に機能しません。クロスサイト・スクリプティング攻撃の多くは、悪意ある人が用意した罠(メールのリンクや罠ページ等)から、脆弱なウェブアプリケーションに直接リクエストされるため、本例のような入力側のチェック機構を経由することがないためです。

また、クロスサイト・スクリプティングの脆弱性への対策における「入力チェック」は、そもそも漏れが生じやすく、根本的な対策にはなりません。1章5節「クロスサイト・スクリプティング」の根本的対策の内容を参考に、対策を検討してください。

■ ブラックリスト方式による入力チェックのみを実装

【脆弱な実装】

```
if ($a =~ /(script|expression|...)/i) { # 入力チェック
    error_html(); # 危険な値を含む場合はエラー表示
    exit;
} else {
    ...
    print $a; # 危険な値を含まない場合は処理を先に進める
```

Perl

本例は、ブラックリスト方式による入力チェック機構を実装しているウェブアプリケーションです。ブラックリストには、クロスサイト・スクリプティング攻撃に悪用される危険な文字列を定義しています。たとえば、入力値 \$a に「script」等を含む場合、処理を先に進めず、エラー画面を表示します。

一見、入力チェック機構が有効に機能し、クロスサイト・スクリプティング攻撃を無効化できるように思われますが、この実装にはチェックを回避され、攻撃が成功してしまう問題が存在します。

【解説】

制御文字等を悪用した入力チェックの回避

「入力チェック」は、クロスサイト・スクリプティングの脆弱性への根本的な対策にはなりません。たとえば、\$a に、下記のような文字列を指定された場合、入力チェックによる「script」のマッチングを回避されます。

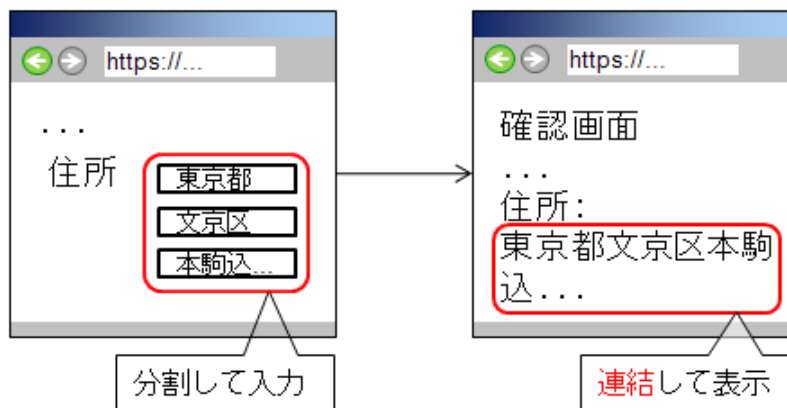
```
<s%00cript>alert(0)</s%00cript>
```

テキスト

入力チェックを通過した \$a は、「%00」がデコードされた Null 文字を含む形でウェブページに出力されます。ウェブブラウザによってはこの Null 文字を無視するため、結果として \$a の出力内容はスクリプト文字列として解釈されます。このため、単純なパターンマッチングでは、スクリプトに悪用される文字列を完全に抽出することはできません。Null 文字のような、チェック機構の回避に悪用可能な文字は、他にも複数存在します。

【よくある失敗例 1】

連結処理を悪用した入力チェックの回避



本例は、ブラックリスト方式による入力チェック機構のみを実装しているウェブアプリケーションにおける問題です。入力フォームには、住所を都道府県や市区町村単位に区切って登録する項目が用意されています。入力チェックには、ブラックリストに従って、「script」等の文字を含む場合に、処理を終了する仕組みが設けられています。入力チェックを通過した住所情報は、上図のように連結して表示されます。

```

if ($addr1 =~ /script/i) {      # 入力チェック ($addr2, $addr3 も同処理)
    error_html ();            # 危険な値を含む場合はエラー表示
    exit;
} else {
    ...
    print $addr1. $addr2. $addr3; # 入力チェック後の文字を連結

```

Perl

ここで、住所の相当するパラメータに対し、下記文字列が指定された場合を考えます。

変数	値
\$addr1	<scr
\$addr2	ipt>alert(1)</s
\$addr3	cript>

それぞれの要素は「script」に一致する文字列を含んでいません。したがって、これらは入力チェックを通過します。入力チェック通過後、3つの要素は連結され、下記の文字列が形成されます。

```
<script>alert(1)</script>
```

テキスト

これは、仮に前述の制御文字による入力チェックの回避を解消できていた場合でも生じうる問題です。

入力チェック対策は、入力チェック通過後の演算処理の結果がスクリプト文字列を形成してしまう場合等に対処できない、という性質があります。他の根本的対策と併せて活用することをお勧めします。

【よくある失敗例 2】

削除処理を悪用した対策回避

```

$a =~ s/(script|expression|...)//gi;
...
print $a;

```

Perl

本例は、入力値に対し、ブラックリスト内に一致した文字列を削除する実装のウェブアプリケーションです。たとえば、\$a に文字列「script」が含まれていた場合、その文字列を削除した結果を出力します。

ここで、\$a に下記文字列が指定された場合を考えます。

```
<script>alert(1)</script>
```

テキスト

上記文字列は、ブラックリスト方式による削除処理により、下記のとおり出力されます。script タグが無くなるため、攻撃は成立しません。

```
<>alert(1)</>
```

テキスト

しかし、下記文字列が指定された場合を考えてみます。

```
<script>alert(1)</script>
```

テキスト

上記文字列は、削除処理により、下記のとおりスクリプト文字列を形成して出力されます。

```
<script>alert(1)</script>
```

HTML

このように、危険な文字列を単純に削除する処理は、その結果がスクリプト文字列を形成してしまう可能性があるため、お勧めできません。危険な文字を排除したい場合は、削除ではなく、無害な文字列へ置換することをお勧めします。

詳細は 1.5.2 保険的対策 5-(vii) の内容を参照してください。

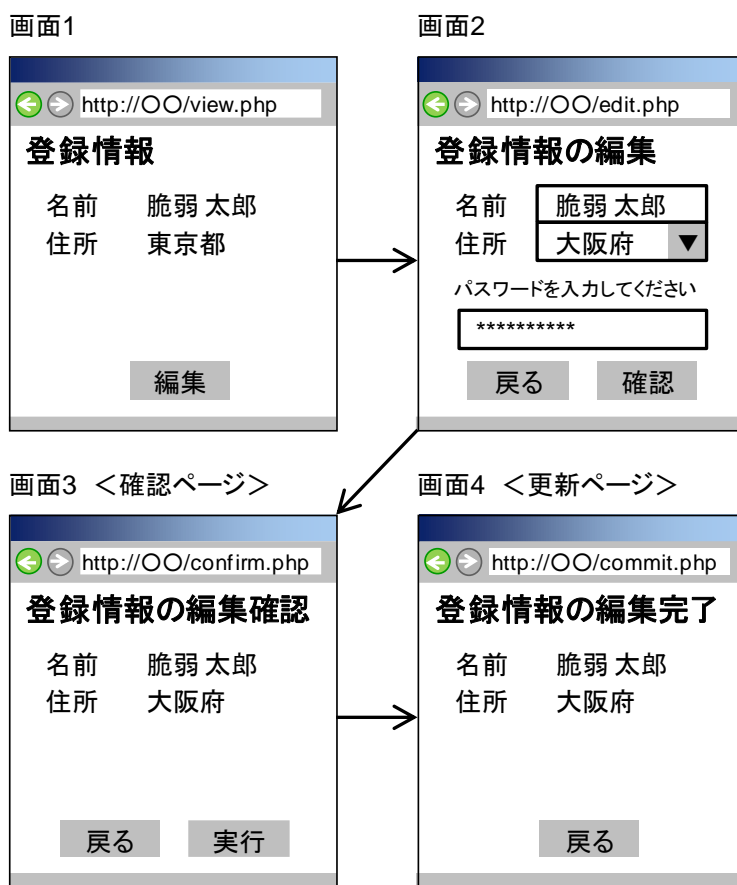
3.6 CSRF(クロスサイト・リクエスト・フォージェリ)の例

CSRF(クロスサイト・リクエスト・フォージェリ)の脆弱性を考慮できていない例として、登録情報編集画面のプログラムを紹介します。

■ PHP による登録情報編集機能

【脆弱な実装】

下図は、会員制ウェブサイトにおける、ユーザの会員登録情報を変更する機能の、典型的な画面遷移の例です。ここでは、ユーザが住所を東京都から大阪府に変更するときの操作を例にしています。



このサイトの構成では、まず画面1(view.php)で登録情報を確認し、編集の必要があれば編集ボタンを押して画面2(edit.php)へ進み、画面2で必要な情報を入力して、さらにパスワードを入力した上で、画面3(confirm.php)へ進むようになっています。画面3で入力した情報を確認し、実行ボタンを押して画面4(commit.php)に進むと、ここで登録情報の更新処理が実行されて、その旨が表示されます。

画面2で、本人確認のためにパスワードを入力するようになっており、パスワードが正しい場合にしか画面3に進めないようになっています。

ここで、画面3を構成するHTMLが次のようになっているとします。

```
<form action="commit.php" method="post">
  <input type="hidden" name="new_name" value="脆弱 太郎">
  <input type="hidden" name="new_address" value="大阪府">
  <input type="submit" name="back" value="戻る">
  <input type="submit" name="commit" value="実行">
</form>
```

HTML

画面 3 でユーザが実行ボタンを押したときに、遷移先である画面 4 の commit.php の次のコードが登録情報の更新処理を実行します。\$_SESSION['authenticated'] は、ユーザがログイン済みかどうかの情報を、真偽値で保持しています。

```
session_start();
if( ! $_SESSION['authenticated'] ) { exit(); }
update_userinfo($_SESSION['uid'], $_POST['new_name'], $_POST['new_address']);
```

PHP

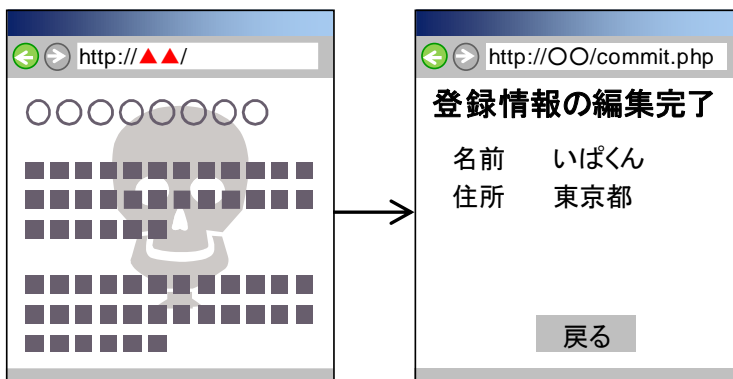
この実装は、2 行目でユーザがログイン済みかどうかを確認していますが、そのリクエストのパラメータがユーザの意図した操作によるものであるかを判別する機能を持っていないため、CSRF 攻撃に対して脆弱です。

【解説】

ユーザがこのサイトにログインしたままとなっているタイミングで、罫サイトに誘導されると、画面 4 にリダイレクトされることによって、ユーザの意図に反した登録情報の変更処理を実行させられてしまいます。

罫サイト

画面4 <更新ページ>



仕掛けの一例として、罫サイトに次のような HTML が含まれる場合があります。これは、画面 3 を構成する HTML に似たものですが、赤字部分が異なります。罫サイトを開くだけで、CSRF 攻撃が実行されます。

```
<form action="http://○○/commit.php" method="post" name="f1">
  <input type="hidden" name="new_name" value="いばくん">
  <input type="hidden" name="new_address" value="東京都">
</form>
<script>document.forms['f1'].submit();</script>
```

HTML

例題コードの commit.php は、上記のような罠サイトによって引き起こされたリクエストと、ユーザの意図した操作によるリクエストを判別できず、登録情報を更新してしまいます。

ログイン済みのユーザが設定変更や書き込みを行う機能を実装する際は、CSRF 攻撃の存在を意識して、対策を検討してください。

【修正例 1】

秘密情報の埋め込みと更新ページにおける確認

確認ページに秘密情報を埋め込んだ上で、更新ページで確認することで、CSRF の脆弱性に対する根本的解決となります。

例題コードにおいては、画面 3 が確認ページに相当し、画面 4 が更新ページに相当します。この修正例では、秘密情報として PHP のセッション ID を使用します。

まず、画面 3 にセッション ID を埋め込みます(赤字部分が該当箇所)。

```
<form action="commit.php" method="post">
  <input type="hidden" name="new_name" value="脆弱 太郎">
  <input type="hidden" name="new_address" value="大阪府">
  <input type="hidden" name="sid" value="6a0752gpmhignmq9f5iah8h71">
  <input type="submit" name="back" value="戻る">
  <input type="submit" name="commit" value="完了">
</form>
```

HTML

次に、画面 4 の commit.php が秘密情報を確認します。秘密情報の確認タイミングは、登録情報を更新する前です。もし秘密情報が正しくない場合、処理を中止します。修正した commit.php は、次のものになります(赤字部分を追加)。

```
session_start();
if( ! $_SESSION['authenticated'] ) { exit(); }
if( $_POST['sid'] != session_id() ) { exit(); }
update_userinfo($_SESSION['uid'], $_POST['new_name'], $_POST['new_address']);
```

PHP

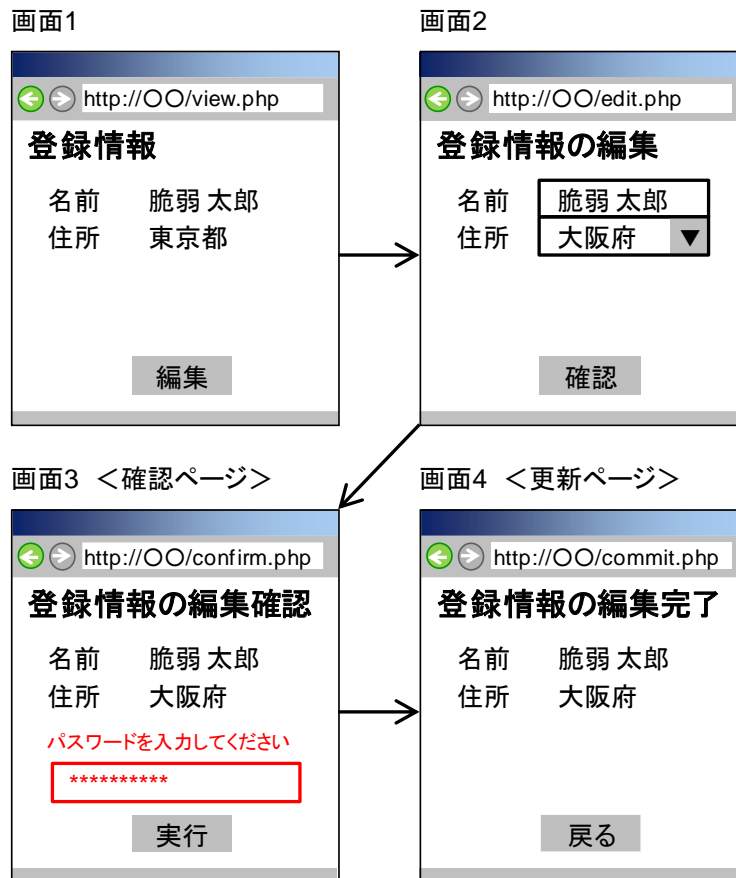
この修正には、第三者に予測されない秘密情報を生成・保持する事と、秘密情報を更新ページに伝達する際に POST メソッドを使用する事が必要です。これらの条件を満たせない場合、別の方法で修正します。

【修正例 2】

更新ページにおけるパスワードの確認

更新ページの直前でパスワードの入力を求め、更新ページでパスワードを確認することで、CSRF の脆弱性に対する根本的解決となります。

例題コードでは、画面 2 で入力されたパスワードを画面 3 で確認していました。これを、画面 3 でパスワードを入力させて画面 4 でそれを確認する方式に変更することで、CSRF の脆弱性を解消できます。



この修正には、ユーザインターフェースの変更を許容する必要があります。それが難しい場合、別の方法で修正します。

【修正例 3】

更新ページにおける Referer の確認

更新ページで Referer を確認することで、CSRF の脆弱性に対する根本的解決となります。この修正例では、例題コードの `commit.php` を次のように修正します(赤字部分を追加しています)。

```
session_start();
if( !_SESSION['authenticated'] ) { exit(); }
if( $_SERVER['HTTP_REFERER'] != 'http://〇〇/confirm.php' ) { exit(); }
update_userinfo($_SESSION['uid'], $_POST['new_name'], $_POST['new_address']);
```

PHP

この修正の副作用として、ブラウザが Referer を送出不い設定になっている場合や、ブラウザから送出される Referer をプロキシサーバで削除している環境からの利用では、この変更処理が実行できなくなってしまう。

3.7 HTTP ヘッダ・インジェクションの例

HTTP ヘッダ・インジェクションの脆弱性を考慮できていない例として、リダイレクタのプログラムを紹介します。

■ Perl による URL リダイレクション

【脆弱な実装】

```
$cgi = new CGI;
$num = $cgi->param(' num');
print "Location: http://example.jp/index.cgi?num=$num¥n¥n";
```

Perl

これは Location ヘッダによって、アクセスしたユーザを指定の URL へリダイレクトさせるプログラムの一部です。このプログラムではまず \$num 変数に num パラメータの値を代入します(2 行目)。このプログラムは、\$num 変数を基に Location ヘッダを作成し、HTTP レスポンスとして出力します(3 行目)。この実装では、num パラメータの値が数値であると想定しています。

この実装は num パラメータに改行コードを含む値を指定されることを考慮していないため、想定外の HTTP レスポンスを作成されてしまいます。

【解説】

この実装では num パラメータに「3%0D%0ASet-Cookie:SID=evil」を指定された URL へユーザがアクセスした場合、下記のように攻撃者の意図した Cookie が発行されます。また、num パラメータに指定される文字列によっては、ユーザのブラウザへ偽のページを表示されてしまいます。

```
HTTP/1.x 302 Found
Date: Sat, 07 Mar 2009 01:49:48 GMT
Server: Apache/2.2.3 (Unix)
Set-Cookie: SID=evil
Location: http://example.jp/index.cgi?num=3
Content-Length: 292
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

HTTP レスポンス

【修正例】

ヘッダに埋め込む文字列に改行コードを許可しない

改行コードを許可しないよう、開発者が適切な処理を実装することで、HTTP ヘッダ・インジェクションの脆弱性に対する根本的解決となります。

```

### 複数行にわたる文字列から最初の行を返す関数
# 引数: 文字列。2 つ目以降の引数は無視する
# 戻り値: 改行コード (¥r, ¥n, ¥r¥n) 以前の文字列。
sub first_line {
    $str = shift;
    return ($str =~ /^([^\r\n]*)/)[0];
}

```

Perl

上記は、引数で与えられた文字列の最初の行(改行なし)を返す関数です。外部から入力されるパラメータの値を出力する場合でも、この関数を通すことで、HTTP レスポンスヘッダのフィールド値として適切な形式となり、脆弱性を解消できます。

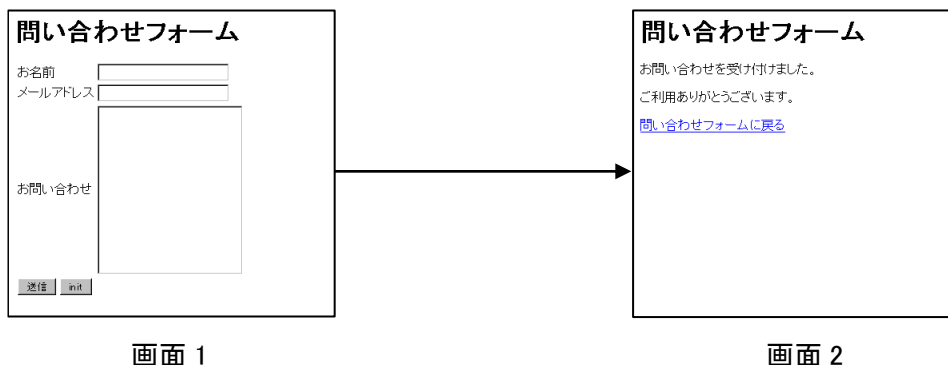
なお、HTTP ヘッダではフィールド値として複数行にわたる値も許可されていますが、この関数は複数行を考慮しない仕様です。複数行にわたる HTTP ヘッダのフィールド値をウェブアプリケーションで扱う場合に、この関数を使用すると、正常なフィールド値を削除してしまいますので注意してください。

3.8 メールヘッダ・インジェクションの例

本節ではメールヘッダ・インジェクションの脆弱性を考慮できていない例として、問い合わせフォームのプログラムを紹介します。

■ Perl によるメール送信機能

【脆弱な実装】



```

open (MAIL, "| /usr/sbin/sendmail -t -i");
print MAIL << "EOF";
To: info¥@example.com
From: $email
Subject: お問い合わせ($name)
Content-Type: text/plain; charset="ISO-2022-JP"

$inquiry
EOF
close (MAIL);

```

Perl

これは、ユーザからの問い合わせ内容をウェブサイト運営者にメールで送信する画面とプログラムの一部⁶⁴です。ユーザが画面 1 における入力欄「お名前」、「メールアドレス」、「お問い合わせ」に値を入力し[送信]ボタンを押すと、プログラムは OS の sendmail コマンドを呼び出して、ウェブサイト運営者のメールアドレス info@example.com にメールを送信します。プログラムがメールを送信する際、入力欄の値をそれぞれ\$name, \$email, \$inquiry 変数に格納し、これらの変数からメールヘッダおよび本文を作成しています。メールの送信を完了すると、画面 2 を出力します。

この実装では、ユーザの入力値をメールヘッダに出力しているため、メールヘッダ・インジェクションの脆弱性があります。

【解説】

この実装では、sendmail コマンドの標準入力にメールヘッダおよびメール本文を与えることでメールを送信します。sendmail コマンドは入力された To ヘッダ、Cc ヘッダ、Bcc ヘッダに基づいて送信先メールアドレスを決定します。ユーザが画面 1 において「お名前」に「anzen」、「メールアドレス」に「anzen@example.net」、「お問い合わせ」に「Hello, World」という値を入力した場合、このプログラムは次のメールを info@example.com に送信します。

```
To: info@example.com
From: anzen@example.net
Subject: お問い合わせ(anzen)
Content-Type: text/plain; charset="ISO-2022-JP"

Hello, World
```

メール

しかし、ユーザが「お名前」または「メールアドレス」に改行コードとメールヘッダを含む値を入力すると、任意のメールアドレスにメールを送信できてしまいます。例えば、このプログラムにユーザが「メールアドレス」として「anzen@example.net%0d%0aBcc%3a%20user@example.org」を入力した場合、sendmail コマンドへの入力は下記ようになります。sendmail コマンドはこの入力に基づいて、info@example.com、の他に user@example.org にもメールを送信してしまいます。

```
To: info@example.com
From: anzen@example.net
Bcc: user@example.org
Subject: お問い合わせ(anzen)
Content-Type: text/plain; charset="ISO-2022-JP"

Hello, World
```

テキスト

【修正例 1】

ユーザの入力値をメールヘッダに出力しない

ユーザの入力値をメールヘッダに出力しないことで、メールヘッダ・インジェクションの脆弱性への根本的解決となります。

⁶⁴ メールヘッダに US-ASCII 以外の文字集合を使用する場合、RFC2047 に基づいて符号化する必要がありますが、この例では省略しました。このため、プログラムが出力する Subject ヘッダの値を日本語で表記しています。

```

open (MAIL, "| /usr/sbin/sendmail -t -i");
print MAIL << "EOF";
To: info¥@example.com
From: webform¥@exmaple.com
Subject: お問い合わせ
Content-Type: text/plain; charset="ISO-2022-JP"

=====
お名前: $name
メールアドレス: $email
=====
$inquiry
EOF

```

Perl

この修正例では、\$name, \$email 変数をメールヘッダに出力せず、メール本文に出力します。From ヘッダ、Subject ヘッダでは、それぞれ「webform@exmaple.com」、「お問い合わせ」という固定値を指定しています。\$name, \$email 変数に改行コードが含まれた場合、メール本文の体裁が崩れますが、任意のメールヘッダを挿入されることはありません⁶⁵。

【修正例 2】

メールヘッダに展開する変数から改行コードを除去する

メールヘッダに展開する変数から改行コードを除去することで、メールヘッダ・インジェクションの脆弱性への保険的対策となります。

```

$name =~ s/¥r|¥n//g;
$email =~ s/¥r|¥n//g;

```

Perl

この修正例では、正規表現を使用してメールヘッダに出力する\$name, \$email 変数から改行コード(「¥r」および「¥n」)を除去しています。

⁶⁵ なお、問い合わせがあったとき、問い合わせ者に自動的に返信する仕様のプログラムの場合、この修正例を採用したとしても、迷惑メール送信に悪用されることがあります。

おわりに

本書で取り上げたウェブアプリケーションのセキュリティ実装やウェブサイトの安全性向上のための取り組みにより、ウェブサイト運営上の脅威の低減が期待できます。また、組織内部でセキュリティ対策の確認、実施を行った上で、外部組織によるペネトレーションテストやウェブアプリケーションのコードチェック等の監査を受けることは、セキュリティ上、より効果的です。ウェブサイトの重要度に応じて、脆弱性検査を受けることをお勧めします。

本書が、ウェブサイトのセキュリティ問題を解決する一助となれば幸いです。本書を執筆するにあたり、発見者や開発者自身から届け出られる脆弱性関連情報を参考にしています。本枠組みにご協力いただいている発見者や開発者の方々に感謝いたします。

用語集

ウェブアプリケーション

ウェブサイトで稼動するシステム。一般に、Java, ASP, PHP, Perl 等の言語を利用して開発され、サイトを訪れた利用者に対して動的なページの提供を実現している。

エスケープ処理

処理系によって特別な意味を持つ文字（記号文字等）に対し、別の文字に置換する等して、特別な意味を持たない文字に変換する処理。

エンコード

データに対し、一定の規則に基づいて符号（コード）化する処理。例えば、URL に利用できない文字（日本語等）は、RFC2396 に基づき、“%” と 16 進数の文字コードにエンコードしなければならない。

改行コード

テキストにおいて、改行を意味する制御コード。一般に、CR (Carriage Return: 行頭復帰) や LF (Line Feed: 改行)、あるいはその 2 つの組み合わせが改行コードとして利用される。ASCII コード体系では、それぞれ“0x0D” と “0x0A” に配置されている。

シェル

ユーザから入力された文字列を解釈し、他のプログラムの起動や制御を行うプログラム。Windows OS では cmd.exe、UNIX/LINUX では bash や csh 等が「シェル」に相当する。

脆弱性（ぜいじゃくせい）

ウェブアプリケーション等におけるセキュリティ上の弱点。コンピュータ不正アクセスやコンピュータウイルス等により、この弱点が攻撃されることで、そのウェブアプリケーションの本来の機能や性能を損なう原因となり得るもの。また、個人情報等が適切なアクセス制御の下に管理されていない等、ウェブサイト運営者の不適切な運用により、ウェブアプリケーションのセキュリティが維持できなくなっている状態も含む。

セッション管理

ウェブサイトが、一連の操作として複数のリクエストを行う利用者を一意に識別するための仕組み。

ディレクトリ・トラバーサル (Directory Traversal)

「.././」のような相対パスを使用し、システム内の任意ファイルへアクセスする攻撃手法。システム内のディレクトリ間を自由に横断（トラバース）できることが攻撃名称の由来。パス・トラバーサルとも呼ばれる。

デコード

エンコードされたデータを元のデータに復元する処理。

ブラックリスト

ホワイトリストと逆の考えに基づいたフィルタ処理の条件定義。リストに登録された内容に一致する文字列の通過を「禁止」する方式。未知の攻撃パターンを検出できない可能性があり、漏れが生じやすいという性質がある。

ホワイトリスト

フィルタ処理で用いられる条件定義の一つ。リストに登録された内容に一致する文字列の通過を「許可」する方式。未知の攻撃パターンにも有効であり、漏れが生じにくい点で安全性は高いが、実装が難しい場合もある。

Cookie

ウェブサーバとウェブブラウザの間に、ユーザに関する情報やアクセス情報等をやりとりするための仕組み。「クッキー」と呼ぶ。

SQL

リレーショナルデータベース (RDB) において、データベースの操作やデータの定義を行うための問い合わせ言語。SQL 文には、CREATE 文等でデータの定義を行う DDL (Data Definition Language: データ定義言語) や SELECT 文、UPDATE 文、GRANT 文等でデータベース操作やアクセス権限の定義を行う DML (Data Manipulation Language: データ操作言語) 等がある。

チェックリスト

このチェックリストは、本書で挙げたウェブアプリケーションの各脆弱性の対策内容をリスト化したものです。これからチェックを行うウェブアプリケーションに対し、対策の要/不要、対策の実施有無を記録し、セキュリティ実装の対応状況を確認する資料としてご活用ください。

■ チェック方法について

各実施項目の対応状況について、次の3つの項目から適切なものを選択してください。

対応済

対策を実施している場合に選択します。

未対応

対策の実施は必要であるが、何らかの理由により未実施の場合に選択します。

対応不要

そもそも脆弱性が存在しない実装である場合や、すでに他の対策を実施し、対策自体が不要であると判断した場合等に選択します。

■ 注意点

- ウェブアプリケーションの性質によっては、本書で挙げた全ての対策が必要になるわけではありません。また、本書で挙げた対策方法は、あくまで解決策の一例です。本文中にて解説した内容を踏まえ、選択した解決策による影響等を十分に考慮した上で、実施を検討してください。
- 実施項目によっては、「いずれかの対策を実施すればよい」というものや、「挙げられた対策を実施できない場合の代替対策」というものがあります(例:SQL インジェクションの脆弱性に対する根本的解決 1-(i)-a と 1-(i)-b の関係)。このような実施項目については、チェック項目をまとめて一つにしています。このチェック項目の「対応済」のチェックは、実施項目のいずれかを実施した場合にチェックを入れてください。また、採用した実施項目のチェックボックスにチェックを入れてください。
- 「根本的解決」につきましては、「脆弱性の原因を作らない実装」を実現する内容であり、実施することが望まれる内容です。チェックリストでは、「根本的解決」と「保険的対策」とを見た目で区別できるように、「根本的解決」の項目を太字と色で強調しています。

No	脆弱性の種類	対策の性質	チェック	実施項目	解説
1	SQLインジェクション	根本的解決	※ <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/> SQL文の組み立ては全てプレースホルダで実装する。	1-(i)-a
				<input type="checkbox"/> SQL文の構成を文字列連結により行う場合は、アプリケーションの変数をSQL文のリテラルとして正しく構成する。	1-(i)-b
		根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	ウェブアプリケーションに渡されるパラメータにSQL文を直接指定しない。	1-(ii)
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	エラーメッセージをそのままブラウザに表示しない。	1-(iii)
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	データベースアカウントに適切な権限を与える。	1-(iv)
2	OSコマンド・インジェクション	根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/> シェルを起動できる言語機能の利用を避ける。	2-(i)
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	シェルを起動できる言語機能を利用する場合は、その引数を構成する全ての変数に対してチェックを行い、あらかじめ許可した処理のみを実行する。	2-(ii)
3	パス名パラメータの未チェック/ ディレクトリ・トラバーサル	根本的解決	※ <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/> 外部からのパラメータでウェブサーバ内のファイル名を直接指定する実装を避ける。	3-(i)-a
				<input type="checkbox"/> ファイルを開く際は、固定のディレクトリを指定し、かつファイル名にディレクトリ名が含まれないようにする。	3-(i)-b
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	ウェブサーバ内のファイルへのアクセス権限の設定を正しく管理する。	3-(ii)
4	セッション管理の不備	根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	セッションIDを推測が困難なものにする。	4-(i)
		根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	セッションIDをURLパラメータに格納しない。	4-(ii)
		根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	HTTPS通信で利用するCookieにはsecure属性を加える。	4-(iii)
		根本的解決	※ <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/> ログイン成功後に、新しくセッションを開始する。	4-(iv)-a
				<input type="checkbox"/> ログイン成功後に、既存のセッションIDとは別に秘密情報を発行し、ページの遷移ごとにその値を確認する。	4-(iv)-b
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	セッションIDを固定値にしない。	4-(v)
保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	セッションIDをCookieにセットする場合、有効期限の設定に注意する。	4-(vi)		
※ このチェック項目の「対応済」のチェックは、実施項目のいずれかを実施した場合にチェックします。					

No	脆弱性の種類	対策の性質	チェック	実施項目	解説		
5	クロスサイト・スクリプティング	根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	ウェブページに出力する全ての要素に対して、エスケープ処理を施す。	5-(i)		
				URLを出力するときは、「http://」や「https://」で始まるURLのみを許可する。	5-(ii)		
				<script>...</script> 要素の内容を動的に生成しない。	5-(iii)		
				スタイルシートを任意のサイトから取り込めるようにしない。	5-(iv)		
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	入力値の内容チェックを行う。	5-(v)		
				根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	入力されたHTMLテキストから構文解析木を作成し、スクリプトを含まない必要な要素のみを抽出する。	5-(vi)
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要			入力されたHTMLテキストから、スクリプトに該当する文字列を排除する。	5-(vii)
				全てのウェブアプリケーションに共通の対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	HTTPレスポンスヘッダのContent-Typeフィールドに文字コード(charset)の指定を行う。	5-(viii)
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要			Cookie情報の漏えい対策として、発行するCookieにHttpOnly属性を加え、TRACEメソッドを無効化する。	5-(ix)
						保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要
CSRF (クロスサイト・リクエスト・フォージェリ)	根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/> 処理を実行するページを POST メソッドでアクセスするようにし、その「hidden パラメータ」に秘密情報が挿入されるよう、前のページを自動生成して、実行ページではその値が正しい場合のみ処理を実行する。				
			<input type="checkbox"/> 処理を実行する直前のページで再度パスワードの入力を求め、実行ページでは、再度入力されたパスワードが正しい場合のみ処理を実行する。	6-(i)-b			
			<input type="checkbox"/> Refererが正しいリンク元かを確認し、正しい場合のみ処理を実行する。	6-(i)-c			
			<input type="checkbox"/> 重要な操作を行った際に、その旨を登録済みのメールアドレスに自動送信する。	6-(ii)			
7	HTTPヘッダ・インジェクション	根本的解決	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/> ヘッダの出力を直接行わず、ウェブアプリケーションの実行環境や言語に用意されているヘッダ出力用APIを使用する。	7-(i)-a		
				<input type="checkbox"/> 改行コードを適切に処理するヘッダ出力用APIを利用できない場合は、改行を許可しないよう、開発者自身で適切な処理を実装する。	7-(i)-b		
		保険的対策	<input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	外部からの入力の全てについて、改行コードを削除する。	7-(ii)		

※ このチェック項目の「対応済」のチェックは、実施項目のいずれかを実施した場合にチェックします。

No	脆弱性の種類	対策の性質	チェック	実施項目	解説
8	メールヘッダ・インジェクション	根本的解決 ※ <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/>	メールヘッダを固定値にして、外部からの入力はずべてメール本文に出力する。	8-(i)-a
				ウェブアプリケーションの実行環境や言語に用意されているメール送信用APIを使用する(8-(i)を採用できない場合)。	8-(i)-b
		根本的解決 <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要		HTMLで宛先を指定しない。	8-(ii)
		保険的対策 <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要		外部からの入力の全てについて、改行コードを削除する。	8-(iii)
9	クリックジャッキング	根本的解決 ※ <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/>	HTTPレスポンスヘッダに、X-Frame-Optionsヘッダフィールドを出力し、他ドメインのサイトからのframe要素やiframe要素による読み込みを制限する。	9-(i)-a
				処理を実行する直前のページで再度パスワードの入力を求め、実行ページでは、再度入力されたパスワードが正しい場合のみ処理を実行する。	9-(i)-b
		保険的対策 <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要		重要な処理は、一連の操作をマウスのみで実行できないようにする。	9-(ii)
10	バッファオーバーフロー	根本的解決 ※ <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要	<input type="checkbox"/>	直接メモリにアクセスできない言語で記述する。	10-(i)-a
				直接メモリにアクセスできる言語で記述する部分を最小限にする。	10-(i)-b
		根本的解決 <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要		脆弱性が修正されたバージョンのライブラリを使用する。	10-(ii)
11	アクセス制御や認可制御の欠落	根本的解決 <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要		アクセス制御機能による防御措置が必要とされるウェブサイトには、パスワード等の秘密情報の入力を必要とする認証機能を設ける。	11-(i)
		根本的解決 <input type="checkbox"/> 対応済 <input type="checkbox"/> 未対策 <input type="checkbox"/> 対応不要		認証機能に加えて認可制御の処理を実装し、ログイン中の利用者が他人になりすましてアクセスできないようにする。	11-(ii)
※ このチェック項目の「対応済」のチェックは、実施項目のいずれかを実施した場合にチェックします。					

CWE 対応表

共通脆弱性タイプ一覧 CWE(Common Weakness Enumeration)は、多様な脆弱性を識別するための、脆弱性の種類(脆弱性タイプ)の体系です。CWE では脆弱性の種類を脆弱性タイプとして分類し、それぞれに CWE 識別子(CWE-ID)を付与して階層構造で体系化しています。CWE を用いると、ソフトウェア開発者やセキュリティ専門家等に次のようなメリットがあります。

- ソフトウェアのアーキテクチャ、デザイン、コードに内在する脆弱性に関して、共通の言葉で議論できるようになる。
- 脆弱性検査ツール等、ソフトウェアのセキュリティを向上させるための、ツールの標準の評価尺度として使用できる。
- 脆弱性の原因を認識し、脆弱性の低減を行い、再発を防止するための共通の基準として活用できる。

ここでは、本書で採り上げた脆弱性と CWE の対応表を示します。CWE に基づき個別の脆弱性対策を行う場合や、脆弱性対策の網羅性を確認する場合等に、資料としてご活用ください。

■ 参考 URL

IPA: 共通脆弱性タイプ一覧 CWE 概説

<https://www.ipa.go.jp/security/vuln/CWE.html>

No	「安全なウェブサイトの作り方」 脆弱性の種類	CWE Version 1.5 (日本語)	CWE Version 2.8 (英語)
1	SQL インジェクション	SQLインジェクション (CWE-89)	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (CWE-89)
2	OSコマンド・インジェクション	OSコマンド・インジェクション (CWE-78)	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (CWE-78)
3	パス名パラメータの未チェック/ ディレクトリ・トラバーサル	パス・トラバーサル (CWE-22)	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (CWE-22)
4	セッション管理の不備		Use of Insufficiently Random Values (CWE-330)
			Insufficiently Protected Credentials (CWE-522)
			Sensitive Cookie in HTTPS Session Without 'Secure' Attribute (CWE-614)
			Session Fixation (CWE-384)
5	クロスサイト・スクリプティング	クロスサイト・スクリプティング (CWE-79)	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (CWE-79)
6	CSRF (クロスサイト・リクエスト・フォージェリ)	クロスサイト・リクエスト・フォージェリ (CWE-352)	Cross-Site Request Forgery (CSRF) (CWE-352)
7	HTTP ヘッダ・インジェクション		Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting') (CWE-113)
8	メールの第三者中継		Improper Neutralization of CRLF Sequences ('CRLF Injection') (CWE-93)
9	クリックジャッキング		直接対応するCWEはありません
10	バッファオーバーフロー	バッファエラー (CWE-119)	Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119)
11	アクセス制御や認可制御の欠落	認可・権限・アクセス制御 (CWE-264)	Permissions, Privileges, and Access Controls (CWE-264)
		不適切な認証 (CWE-287)	Improper Authentication (CWE-287)

更新履歴

本書の改訂第 7 版における更新履歴です。

更新日	更新内容
2015 年 3 月 12 日	改訂第 7 版 第 1 刷発行
2015 年 3 月 26 日	改訂第 7 版 第 2 刷発行 p.75 の下記 2 箇所のソースコードを修正 ・■ PHP と MySQL の組み合わせ【修正例 2】 mysqli_real_escape_string() を使用した例に差し替え ・■ Perl (DBI を利用)【脆弱な実装】 本書改訂第 6 版の内容に差し戻し
2016 年 1 月 27 日	改訂第 7 版 第 3 刷発行 p.81 の下記 1 箇所の数値を修正 ・■ Perl によるセッション ID の生成【解説】 セッション ID の数値を修正
2021 年 3 月 31 日	改訂第 7 版 第 4 刷発行 全体的なリンクの見直し、および修正

著作・制作 独立行政法人情報処理推進機構（IPA）

編集責任 金野 千里

執筆者 谷口 隼祐 扇沢 健也 山下 勇太 徳丸 浩
高木 浩光 独立行政法人産業技術総合研究所

協力者 吉岡 浩二 NECシステムテクノロジー株式会社
谷川 哲司 日本電気株式会社
山岸 正 株式会社 日立製作所
藤原 将志 株式会社 日立製作所
草間 正 富士通株式会社
戸塚 紀子 富士通株式会社
伊藤 耕介 若居 和直 大谷 慎吾 永安 佑希允
園田 道夫 大森 雅司 板橋 博之 相馬 基邦
勝海 直人 宮川 寧夫 長谷川 武 木曾田 優
田中 里実 渡辺 貴仁 篠原 崇宏 岡崎 圭輔

※独立行政法人情報処理推進機構の職員については所属組織名を省略しました。所属組織名は執筆当時のものです。

安全なウェブサイトの作り方

－ ウェブアプリケーションのセキュリティ実装とウェブサイトの安全性向上のための取り組み－

[発行] 2006年 1月31日 第1版 第1刷
2006年 5月11日 第1版 第2刷
2006年11月 1日 改訂第2版 第1刷
2007年 3月 1日 改訂第2版 第2刷
2007年 9月10日 改訂第2版 第3刷
2008年 3月 6日 改訂第3版 第1刷
2008年 8月 1日 改訂第3版 第2刷
2010年 1月20日 改訂第4版 第1刷
2010年 8月 5日 改訂第4版 第2刷
2011年 4月 6日 改訂第5版 第1刷
2012年 3月30日 改訂第5版 第2刷
2012年12月26日 改訂第6版 第1刷
2015年 3月12日 改訂第7版 第1刷
2015年 3月26日 改訂第7版 第2刷
2016年 1月27日 改訂第7版 第3刷
2021年 3月31日 改訂第7版 第4刷

[著作・制作] 独立行政法人 情報処理推進機構 技術本部 セキュリティセンター

[協力] 独立行政法人 産業技術総合研究所 情報セキュリティ研究センター

情報セキュリティに関する届出について

IPA セキュリティセンターでは、経済産業省の告示に基づき、コンピュータウイルス・不正アクセス・脆弱性関連情報に関する発見・被害の届出を受け付けています。

ウェブフォームやメールで届出ができます。詳しくは下記のサイトを御覧ください。

URL: <http://www.ipa.go.jp/security/todoke/>

コンピュータウイルス情報

コンピュータウイルスを発見、またはコンピュータウイルスに感染した場合に届け出てください。

不正アクセス情報

ネットワーク(インターネット、LAN、WAN、パソコン通信など)に接続されたコンピュータへの不正アクセスによる被害を受けた場合に届け出てください。

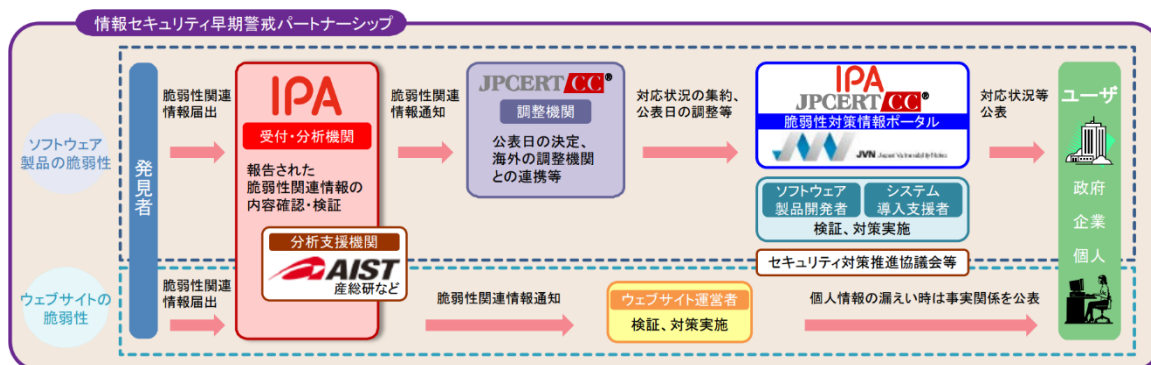
ソフトウェア製品脆弱性関連情報

OSやブラウザ等のクライアント上のソフトウェア、ウェブサーバ等のサーバ上のソフトウェア、プリンタやICカード等のソフトウェアを組み込んだハードウェア等に対する脆弱性を発見した場合に届け出てください。

ウェブアプリケーション脆弱性関連情報

インターネットのウェブサイトなどで、公衆に向けて提供するそのサイト固有のサービスを構成するシステムに対する脆弱性を発見した場合に届け出てください。

脆弱性関連情報流通の基本枠組み「情報セキュリティ早期警戒パートナーシップ」



※IPA:独立行政法人情報処理推進機構, JPCERT/CC:一般社団法人 JPCERTコーディネーションセンター、産総研:国立研究開発法人産業技術総合研究所

IPA

独立行政法人 情報処理推進機構

〒113-6591

東京都文京区本駒込二丁目28番8号

文京グリーンコートセンターオフィス16階

<http://www.ipa.go.jp>

セキュリティセンター

TEL: 03-5978-7527 FAX 03-5978-7518

<http://www.ipa.go.jp/security/>