



INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

見つけられやすい脆弱性と ウェブフレームワークに求められる セキュリティ対策

2019年 8月

独立行政法人情報処理推進機構

セキュリティセンター セキュリティ対策推進部

熊谷 悠平

■講演内容

- 【安全なウェブサイトの作り方】を元に11の脆弱性を解説
 - ◆ 脆弱性の内容と対策方法
- フレームワークと共通部品に求められるセキュリティ対策を説明
 - ◆ 既存のフレームワークでも実装された機能

■脆弱性とは

- ウェブサイトやソフトウェア製品の機能や性能を損なう問題箇所
- 放置されると、ウイルスの感染活動やウェブサイトの乗っ取り、情報漏洩等の被害につながる可能性がある

第一章 見つけやすい11の脆弱性

第二章 フレームワークと共通部品に必要な対策

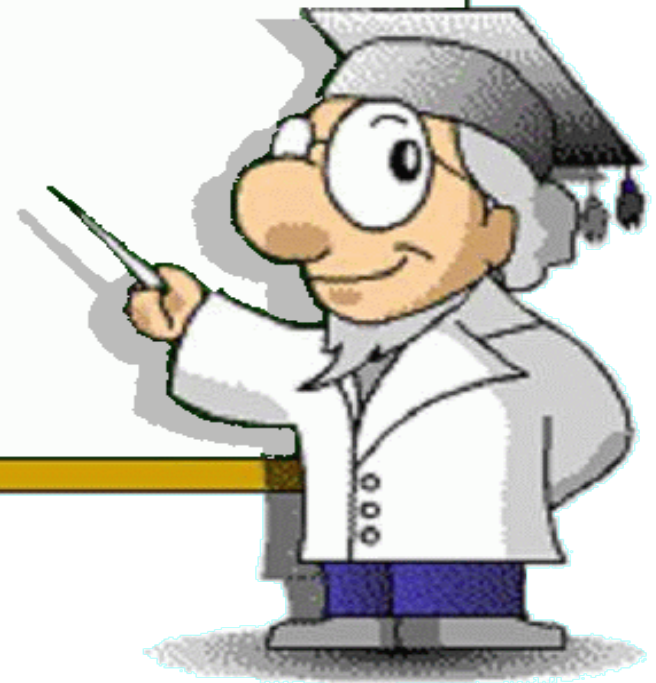


第一章 見つけやすい11の脆弱性 IPA

1.1 IPAへの届出状況

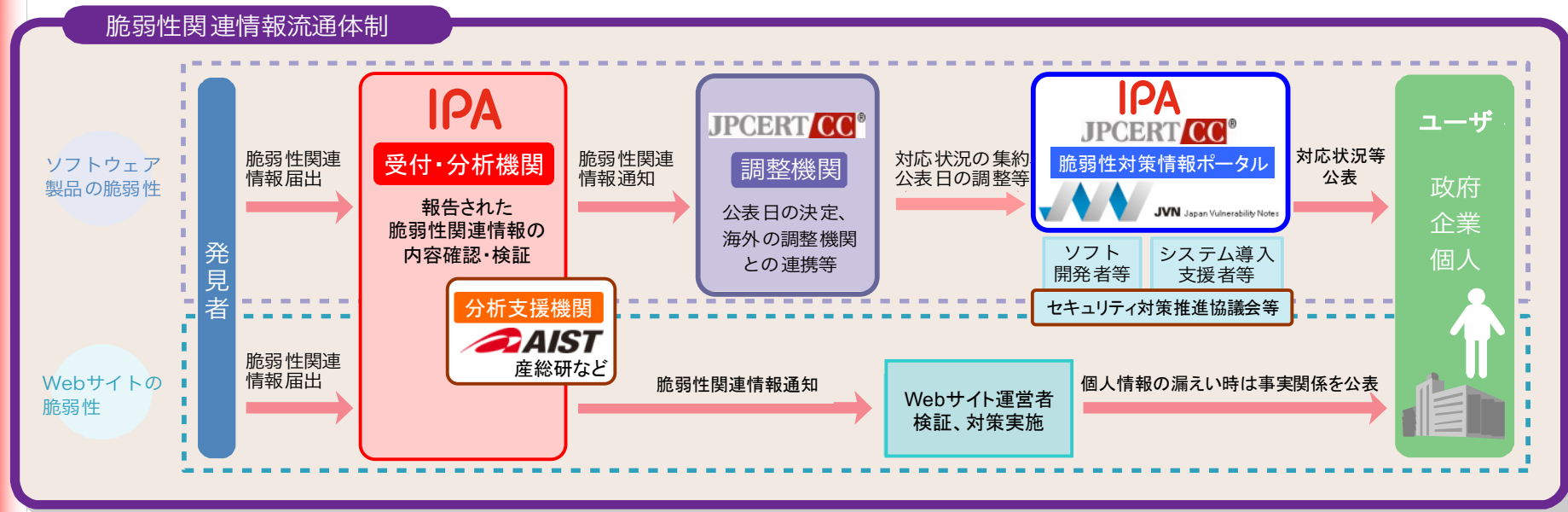
1.2 11の脆弱性

1.3 注意すべき脆弱性と対策



脆弱性届出制度

脆弱性関連情報流通体制

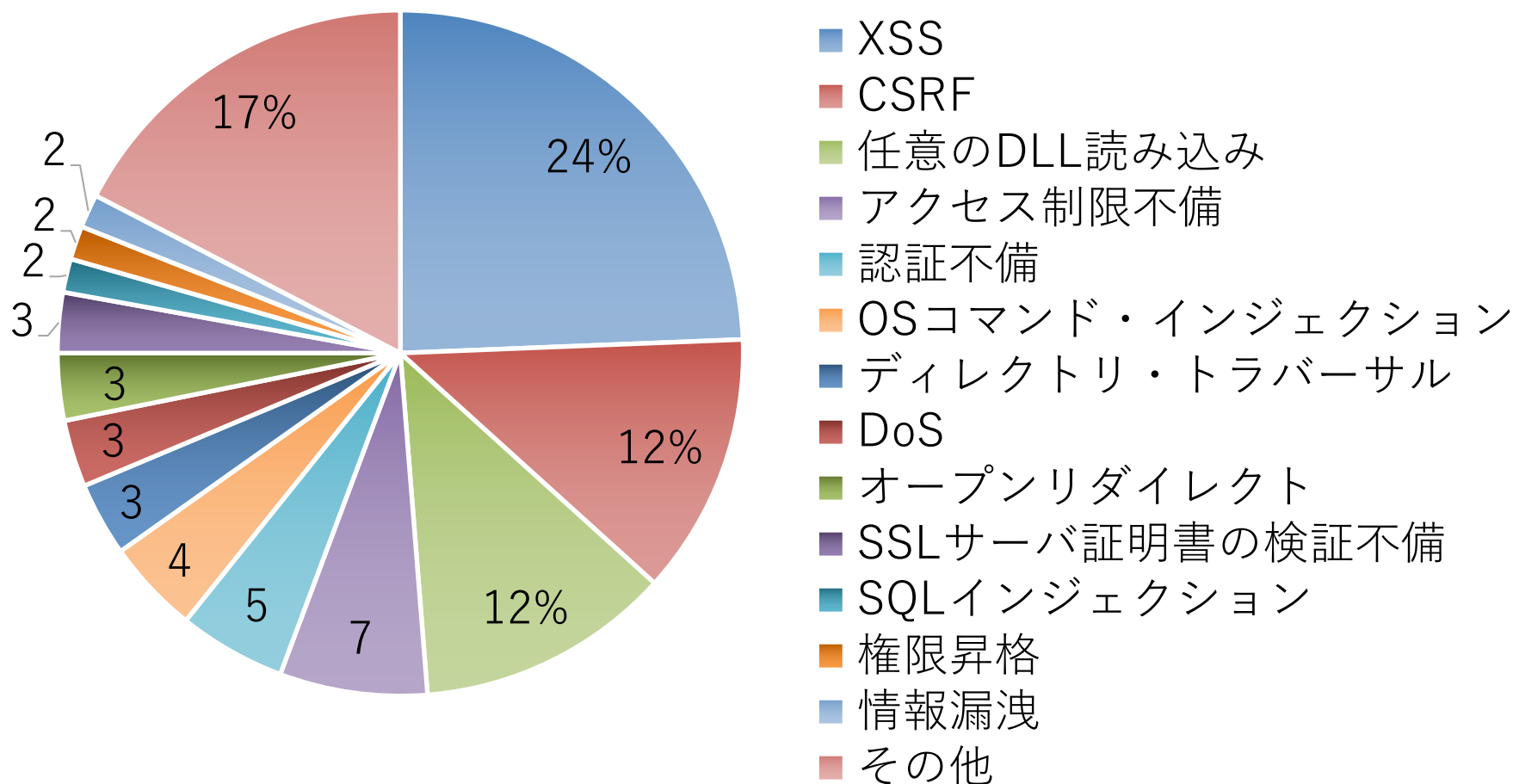


- IPAでは、脆弱性情報を取り扱う「情報セキュリティ早期警戒パートナーシップ」を運営

- 発見者からの届出を受け付け、製品開発者/ウェブサイト運営者に通知し、対策を促す制度

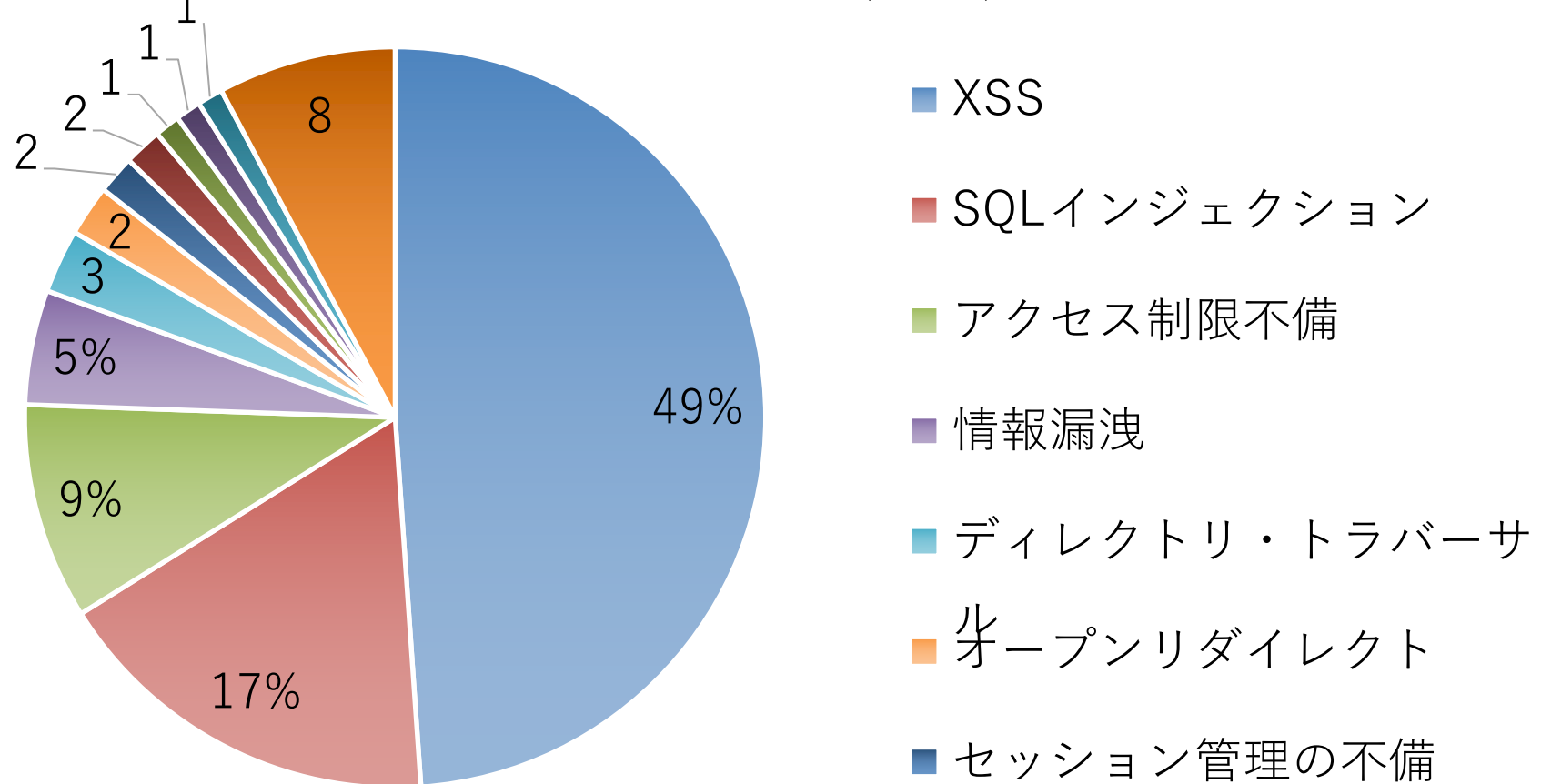
製品に関する脆弱性の届出状況

製品に関する脆弱性の届出件数(2018)



ウェブサイトに関する脆弱性の届出状況IPA

ウェブサイトに関する脆弱性の届出件数(2018)

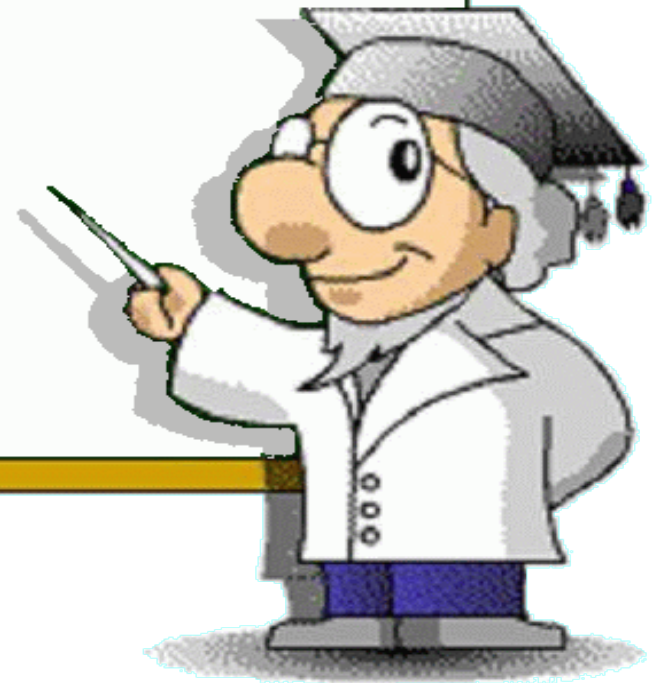


第一章 見つかりやすい11の脆弱性 IPA

1.1 IPAへの届出状況

1.2 11の脆弱性

1.3 注意すべき脆弱性と対策



11の脆弱性

- 安全なウェブサイトの作り方では以下の脆弱性を解説
- IPAに届出が多いものや影響が大きい脆弱性

No.	脆弱性名	ウェブサイトへの 影響が大きい
1	SQLインジェクション	ウェブサイトへの 影響が大きい
2	OSコマンド・インジェクション	
3	ディレクトリ・トラバーサル	
4	セッション管理の不備	見つけられやすい
5	クロスサイト・スクリプティング	
6	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	

第一章 見つけやすい11の脆弱性 IPA

1.1 11の脆弱性

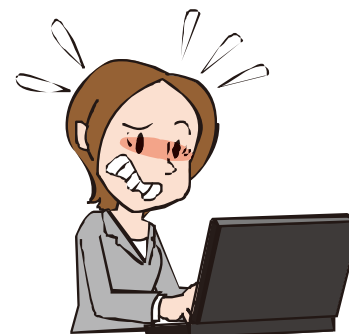
1.2 IPAへの届出状況

1.3 注意すべき脆弱性と対策



□ 影響が大きい物

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. ディレクトリ・トラバーサル



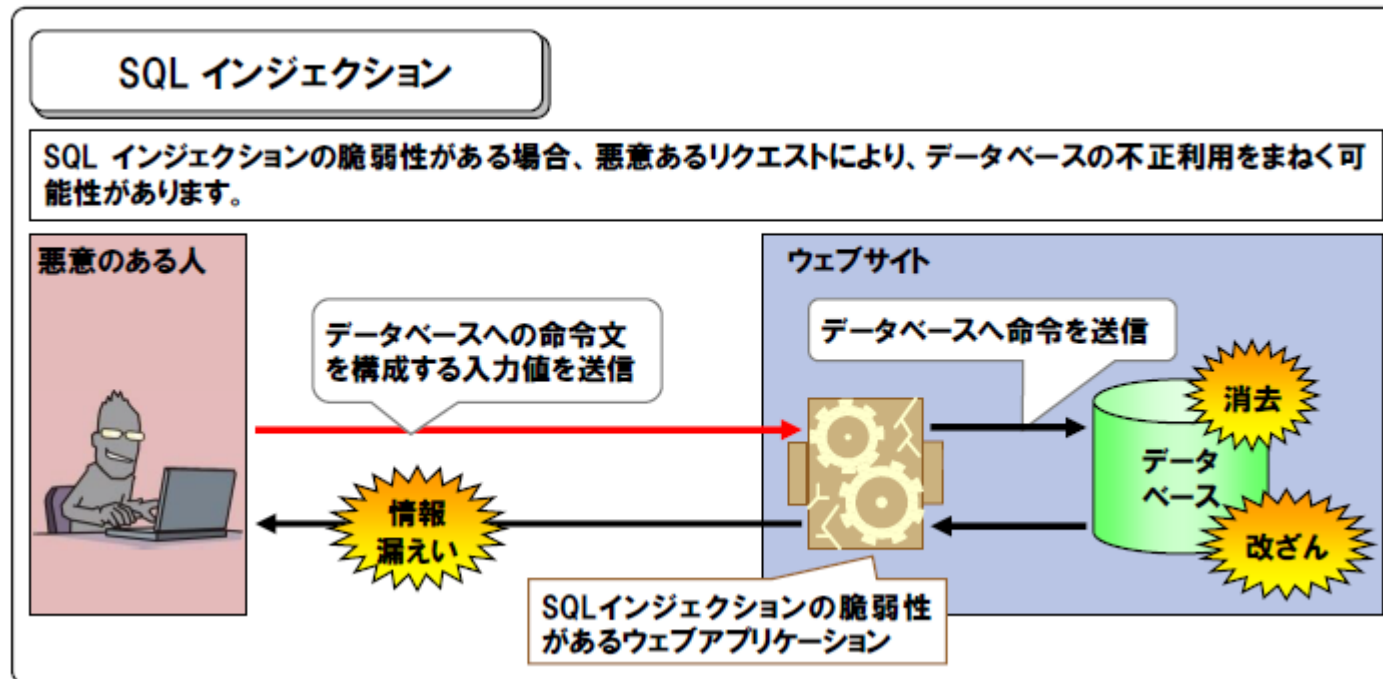
□ 見つけられやすい物

1. クロスサイト・スクリプティング
2. クロスサイト・リクエスト・フォージェリ
3. アクセス制御の欠落



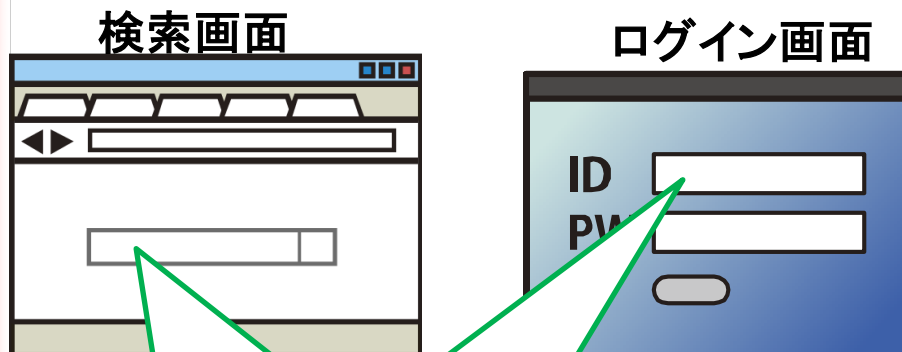
上記や安全なウェブサイトの作り方で
紹介した脆弱性が全てではない

1. SQLインジェクション



- ◆ SQL文の構成に問題がある場合、不正なSQL文が入力され、攻撃者の任意の処理を実行させられる可能性がある
- ◆ 入力値のエスケープ処理に不備があることが原因

1. SQLインジェクション 実例



DBへのリクエストが生じる
箇所に存在

● SQL文の設計
SELECT ~ WHERE uid = '\$uid';

● 入力例
1' or '1'='1

常に「真」

SELECT ~ WHERE uid = '1' or '1'='1';

SQL文に入力値が入り込む際に特殊記号が含まれないようにする。

→ 言語やDBのライブラリを使用して適切にエスケープする必要がある

過去にはエスケープ回避のため、Shift-JIS
やbase64エンコードを使用した例も

【過去事例】

・Shift-JIS

1 0x27 or 0x27 1 0x270x3d0x27 1

・Base64

MSdvcicxJz0nMQ==

● 根本的解決策

- SQL文の組み立ては全てプレースホルダで実装する

→あらかじめSQL構文を設定しておく方式。

入力値はデータベースやライブラリの処理上、「値」として処理される

例:

```
$result = pg_prepare($conn, "query", 'SELECT * FROM usr WHERE name = $1');  
$result = pg_execute($conn, "query", array("Taro"));
```

- 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

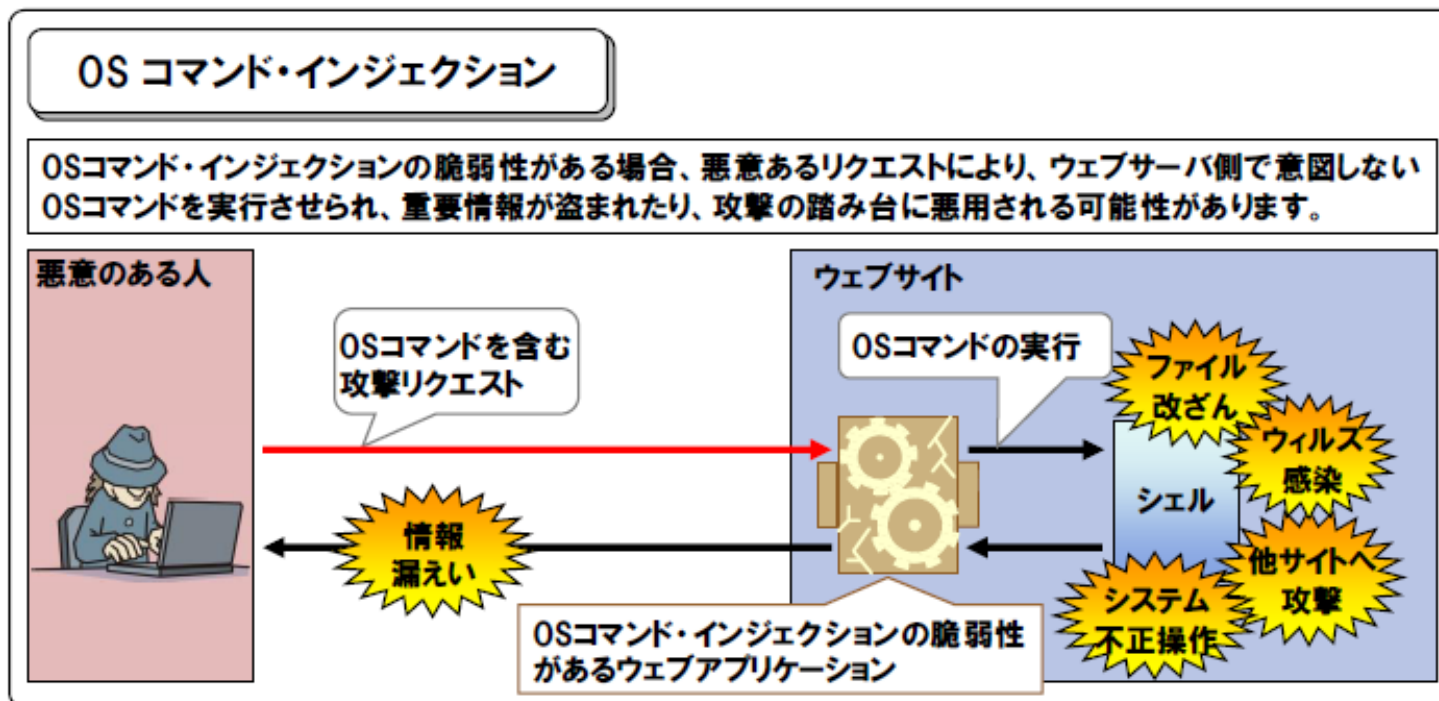
→SQL文を生成する際、文字列連結を使用する場合はエスケープ関数を使用して無害化する

→データベースにより、エスケープが必要な記号が異なるため注意が必要

例:

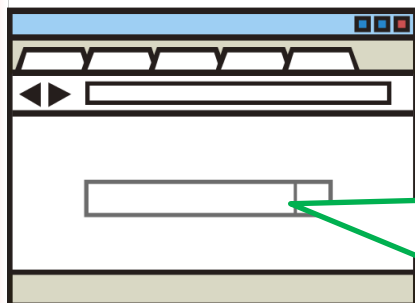
```
$query = "SELECT * FROM usr WHERE name = pg_escape_string($uid)"
```

2.OSコマンド・インジェクション



- ◆ OSの機能を利用する関数を使用しており、外部からの入力値を引数として使用している際、サーバのOSに不正な命令を実行させられる可能性がある
- ◆ シェルを実行できる関数(言語機能)が使用されていることが原因

2. OSコマンドインジェクション 実例



- ・ファイルアップロード
 - ・ファイル参照
 - ・メールフォーム
- 等、OSの機能を利用する箇所が存在

exec等のシェルコマンド実行を行う関数に不正な値が渡されるのが問題

例:

PHP: exec, system

Perl: eval, open, system

入力例: (以下はメールアドレス欄を想定)

test@test.com|\$(cat /etc/passwd | /usr/sbin/sendmail test@test.com)

【過去事例】

- ・ アップロードファイル名からのインジェクション

→zipファイルを解凍する際に、execを使用していたため、**ファイル名を細工**することでOSコマンドが実行されてしまう

例: 「hoge&ping -n 1 localhost&.zip」

- ・ Base64等へのエンコードによる検査回避

例: file = `data:text/html;base64,PHNjcmlwdD5~`

● 根本的解決策

- シェルを起動できる言語機能の利用を避ける

→外部からの入力値を引数として使用する箇所では、exec関数やopen関数等を使用しない。

● 保険的対策

- 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すべ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実行する

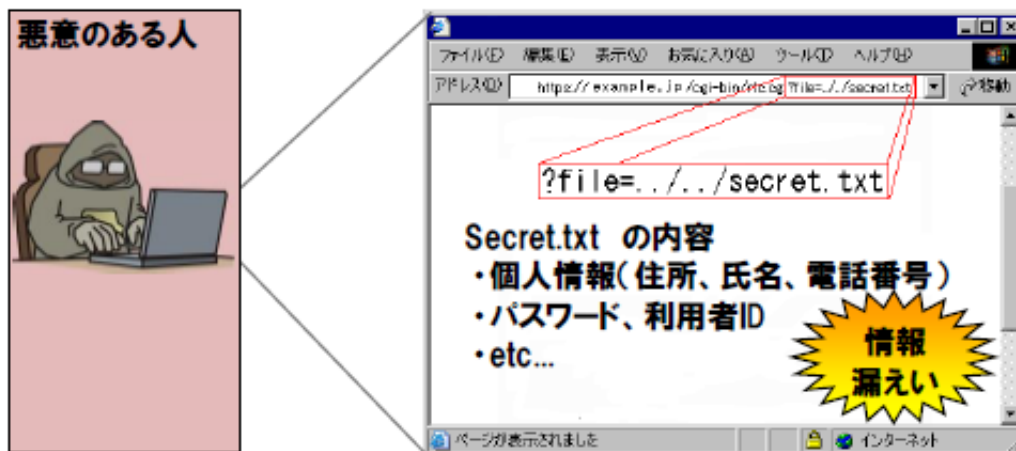
→exec関数やopen関数を使用する場合は、入力値に意図しないOSコマンドが含まれていないか、ホワイトリスト方式でチェックを行う。

ブラックリスト方式では制限漏れの可能性があるため、推奨されない

3.パス名パラメータを悪用したファイル参照 (ディレクトリ・トラバーサル)

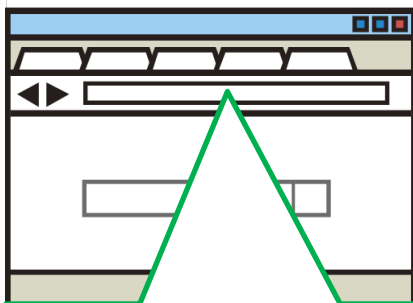
パス名パラメータを悪用したファイル参照

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。



- ◆ 外部からの入力可能なパラメータによってファイル名を指定している場合、意図しないファイルを参照される可能性がある
- ◆ 発生する影響は、ウェブアプリケーションの実装に依存する
- ◆ 特殊記号やディレクトリの指定を許可する実装が問題

3. ディレクトリトラバーサル 実例



- ・URLパラメータでのファイル名指定
- ・ファイル名のハードコーディング等の実装を行っている個所に存在

ファイル名のチェックが不十分であること
によって発生する。

例：

file=../../../../etc/passwd

file=..¥..¥windows¥system32¥net+start|

入力フォームやURLパラメータが
無ければ安全？

POSTメソッドなどでブラウザから送信
する設計であれば、**送信内容を改ざん**
することで、悪用できる場合もある

ディレクトリトラバーサルによる被害内
容は、ウェブアプリケーションの設計に
よる。

→呼び出したファイルをどうするかは
ウェブアプリケーション側の実装に
依存するため。

● 根本的解決

- 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は避ける

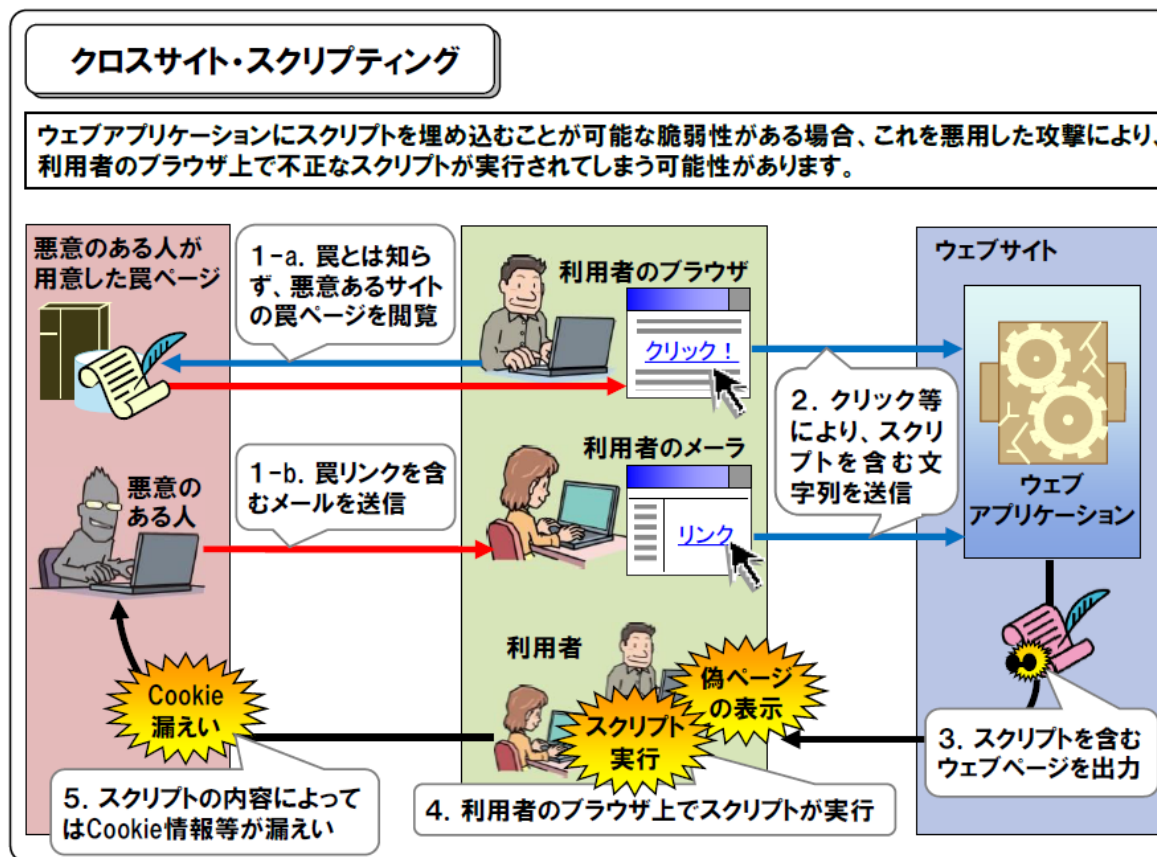
→HTTPリクエストの値でファイルを指定する設計である限り、
ディレクトリトラバーサルの可能性は残る

外部入力を使用したファイル名の指定が必要か、
プログラムの構成を見直す

- 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

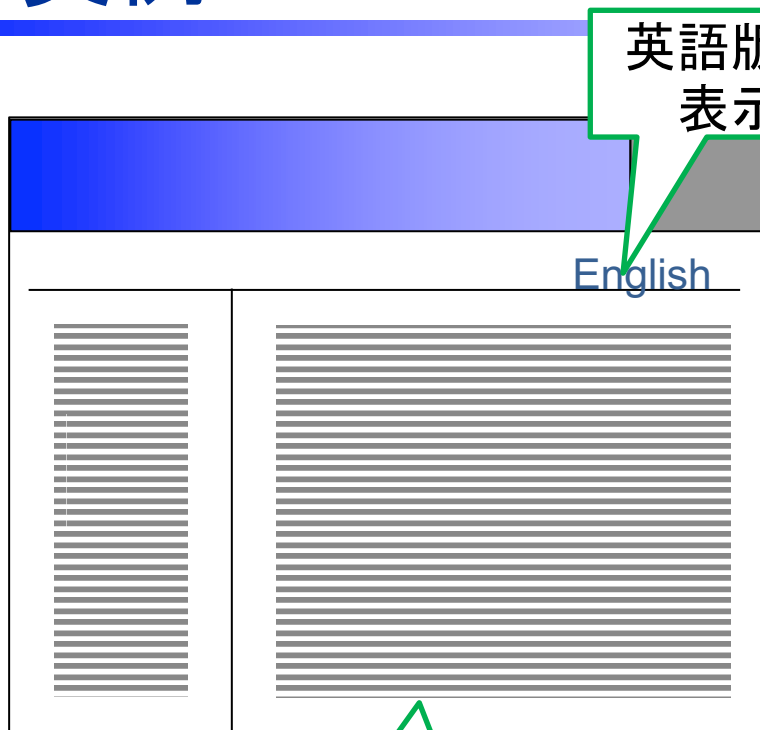
→PHPであれば、basename関数といったファイル名だけを抽出する関数を使用し、ファイルを参照するディレクトリを移動させない実装を行う

4.クロスサイト・スクリプティング



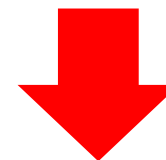
- ◆ 入力値を表示するようなページで、出力する値のエスケープ処理が不十分な場合、意図しないスクリプトを実行させられてしまう可能性がある

4. クロスサイト・スクリプティング 実例



通常のURL:

<https://example.com/report.html>



英語版のURL:

<http://example.com/report.html?lang=english>

英語表示に
切り替わる

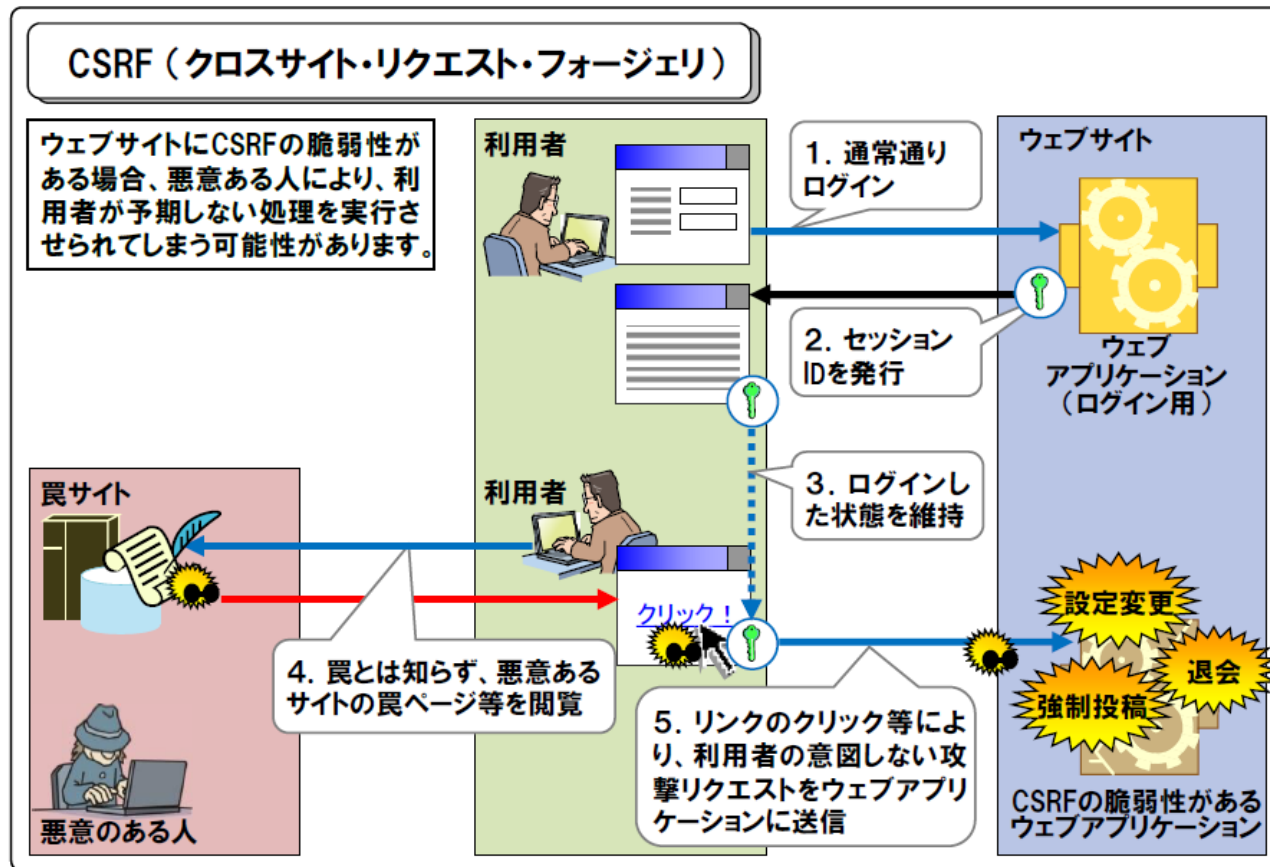
`lang=<script>alert("hoge")</script>`
と送信すると、ポップアップが表示されてしまう

入力欄やページ上で見て取れる動的表示箇所がないからと油断してはいけない

● 根本的解決

- ウェブページに出力するすべての要素に対して、エスケープ処理を施す
 - ウェブページに出力される値に対し、htmlspecialchars関数等を使用し、HTMLタグやスクリプトを構成する記号をエスケープする
- 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
 - 入力値に対して構文解析を行い、ホワイトリスト方式で出力を許可するタグやスクリプト等の要素を抽出する
- 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
 - 文字コードを明示的に指定していないと、ブラウザが文字コードを自動的に判定してしまい、エンコードによりエスケープ処理をすり抜けたスクリプトをブラウザ上で表示してしまう可能性がある

5.クロスサイト・リクエスト・フォージェリ IPA



◆ ウェブサイトにセッション管理の不備等の問題があることで、利用者が意図しない操作を実行させられてしまう可能性がある

実例

- ウェブサイトのセッション管理に不備があることで生じる問題
 - 重要な処理を行う際のセッション管理が不十分
 - 悪意あるページから遷移させられた際、遷移元チェックをしていない
- 意図せずパスワードやメールアドレス等の登録情報の変更等につながる
 - 掲示板等であれば、意図しない投稿をさせられる可能性も
 - クロスサイト・リクエスト・フォージェリを悪用することで、ログインが必要なページの脆弱性の悪用につながることも

● 根本的解決

- 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

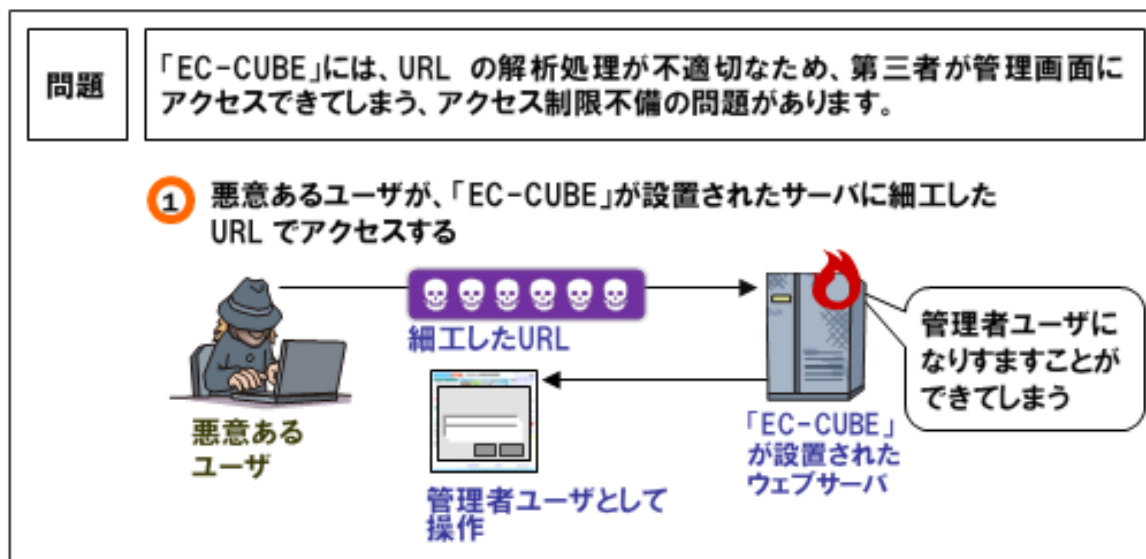
→Cookieによるセッション管理では意図したページからのリクエストであるか判別できない。そのため、ランダムな値のトークンをhiddenパラメータで送信させ、意図しないページからのリクエストでないか判別する

- 処理を実行する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

- 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

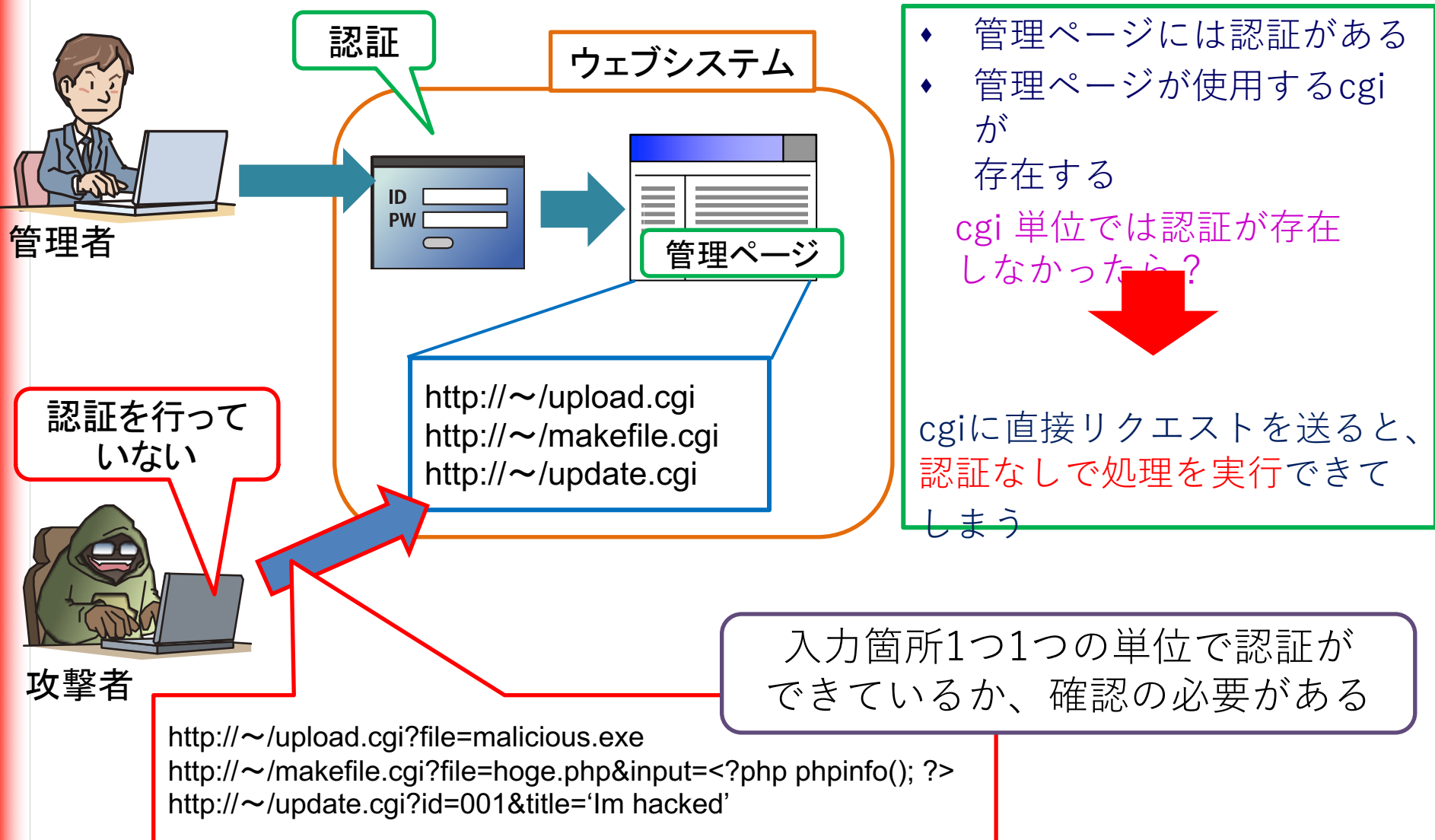
6. アクセス制御や認可制御の欠落

- ウェブサイトのアクセス制御に問題がある場合、権限がないページやファイルを参照できてしまう



- ◆ 認証が必要な機能に認証が存在しないことや、認証を回避できる設計となっている場合、不正に情報を参照されたり、意図せず機能を利用される可能性がある

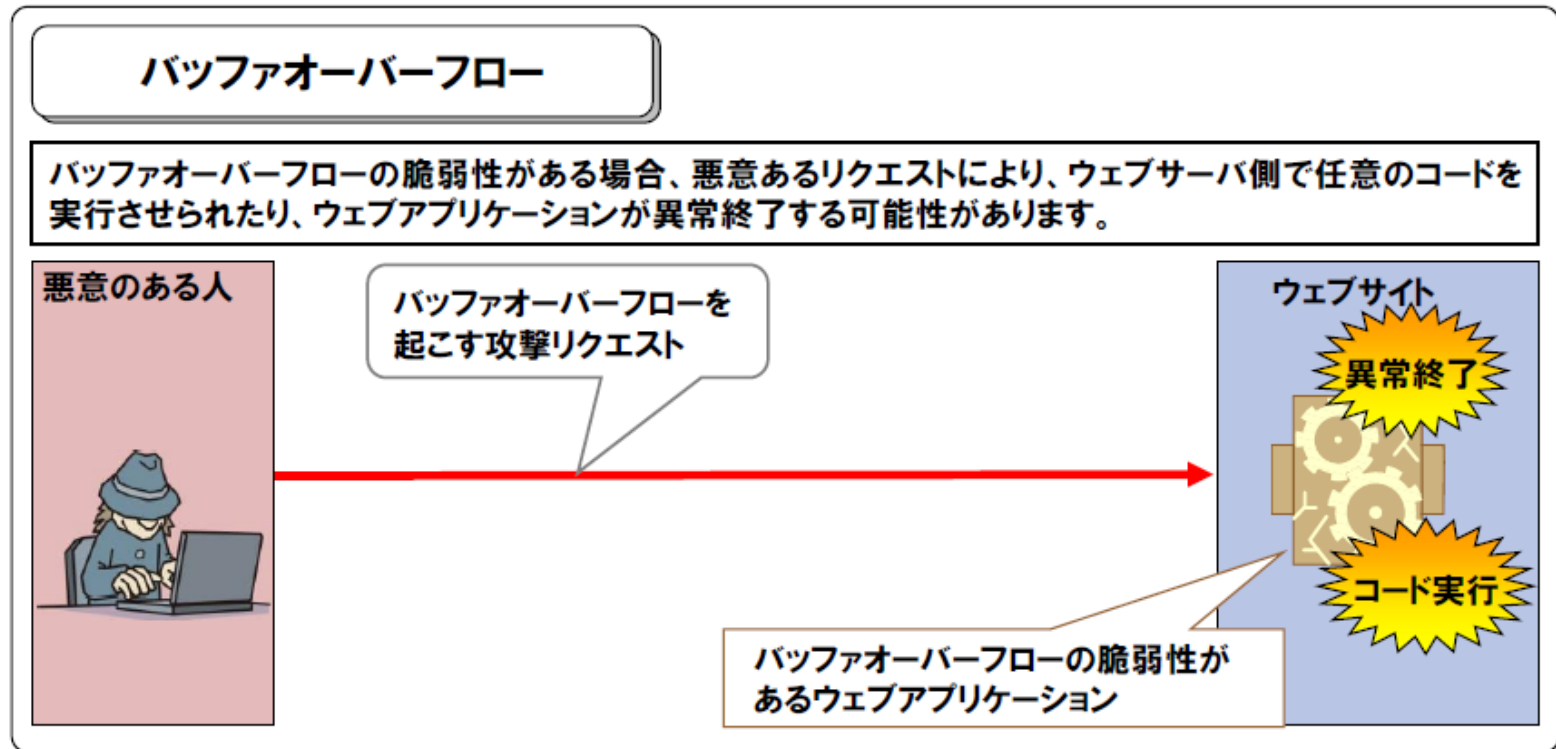
6. クロスサイト・リクエスト・フォージェリ 実例



● 根本的解決

- 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
 - パスワードの入力を受け付ける際は、不必要にログ等に記録されないように注意し、パスワードが必ず暗号化されて保管されるようにする
- 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
 - 認可制御とは「各ユーザにどのような操作を許可するか」の管理
 - セッションIDやトークン等を窃取し、送信するだけで認証を回避できないよう、送信元 IP アドレスとの紐づける等の対策を行う

7. バッファオーバーフロー

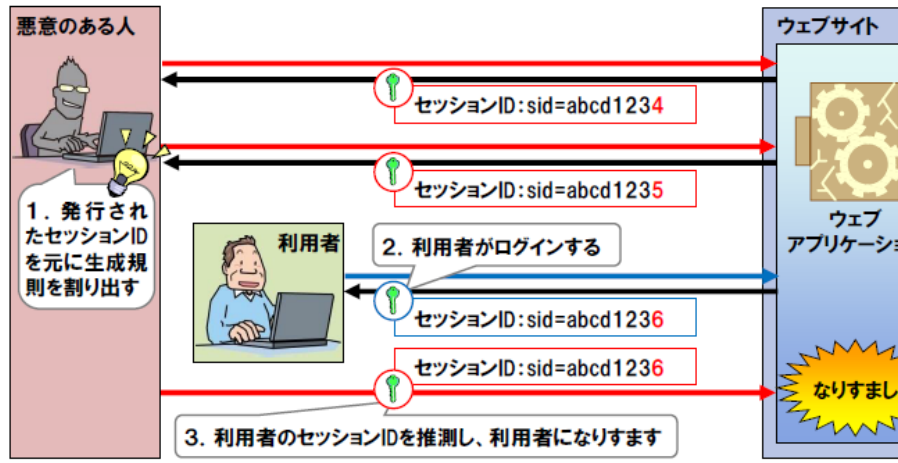


- ◆ プログラムが想定した以上の長さの入力により、確保されたメモリ領域を超えて、入力値がメモリ上に書き出される問題
- ◆ 直接メモリにアクセスできない言語での開発や、入力値の長さを制限するといった対策が必要となる

8.セッション管理の不備

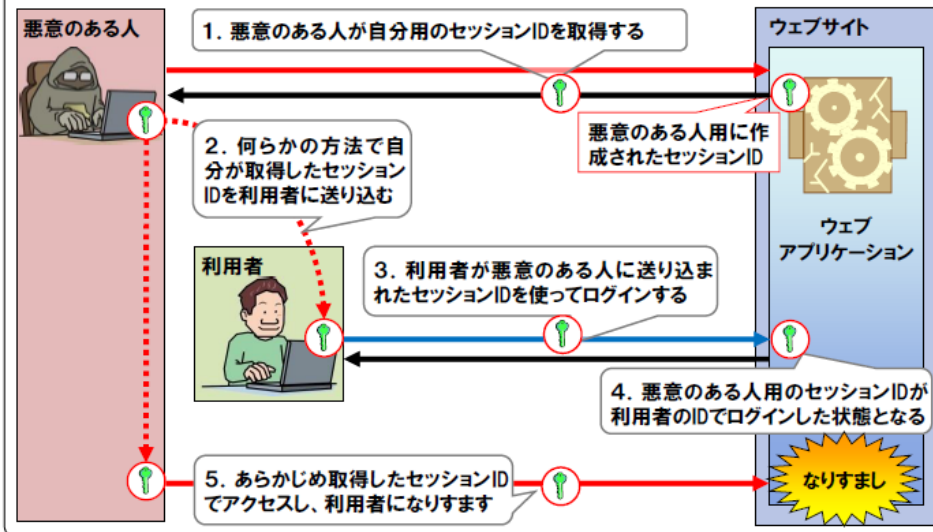
セッションIDの推測

悪意のある人は、セッションIDの生成規則を割り出し、有効なセッションIDを推測します。



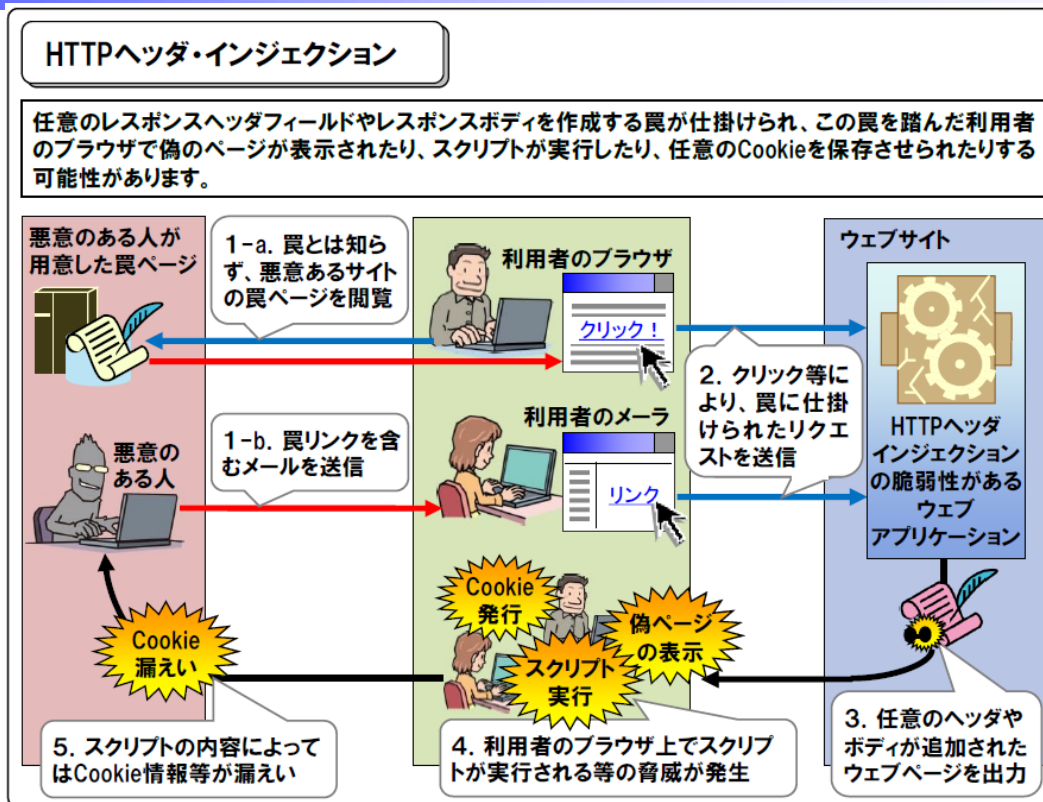
セッションIDの固定化 (Session Fixation)

悪意のある人は何らかの方法で自分が取得したセッションIDを利用者に送り込み、利用者のログインを狙って、その利用者になります。



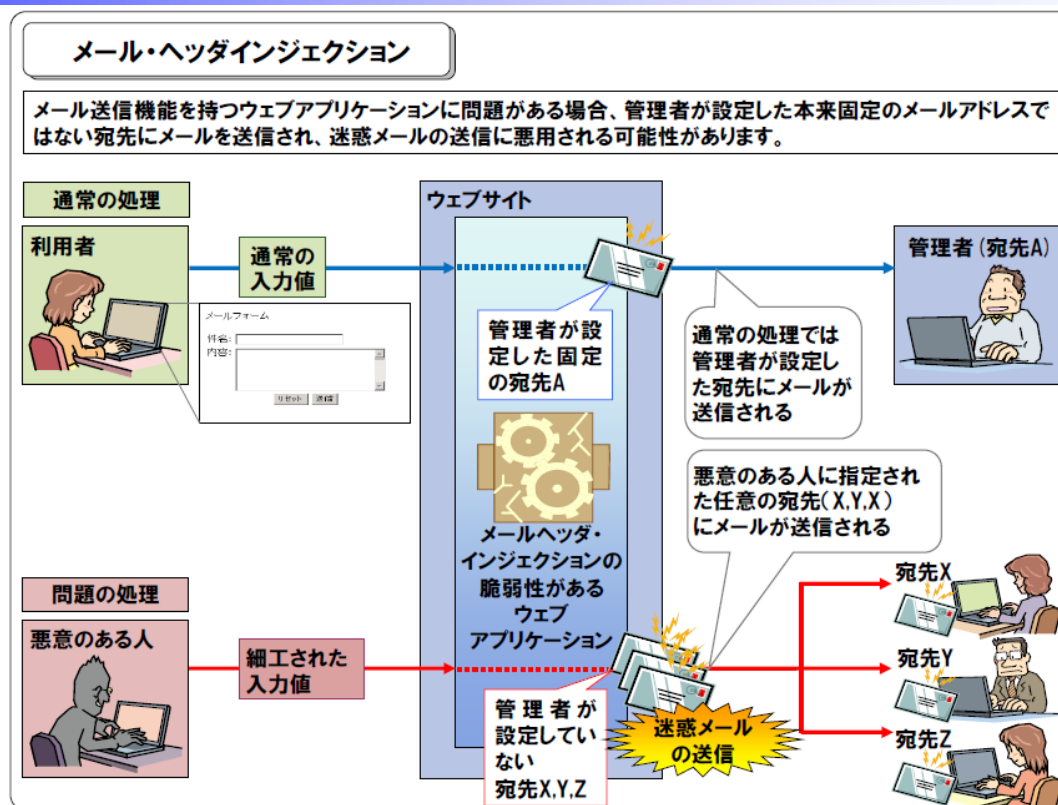
- ◆ セッションIDが推測可能であることや、他者のセッションIDを使用できてしまう場合、第三者に成りすましてログインされる被害の可能性がある。
- ◆ ログインIDを推測や窃取困難にすることや、ログイン後に再発行する、Cookie以外のセッション情報を発行する等の対策が必要となる

9.HTTPヘッダ・インジェクション



- ◆ HTTPヘッダに改行コード等を不正に挿入することで、サーバからのレスポンスを分割し、攻撃者の任意のCookieがキャッシュさせられる等の可能性がある
- ◆ 外部からの入力をそのままヘッダに入力する設計を避けることが必要となる

10.メールヘッダ・インジェクション



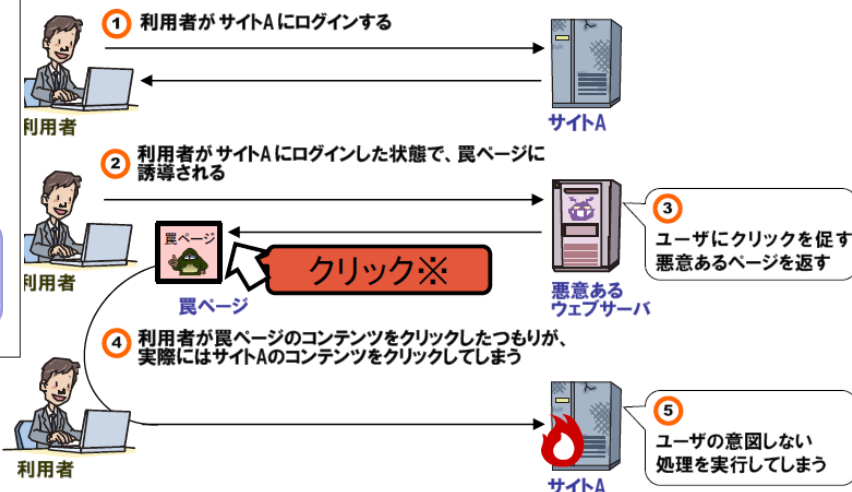
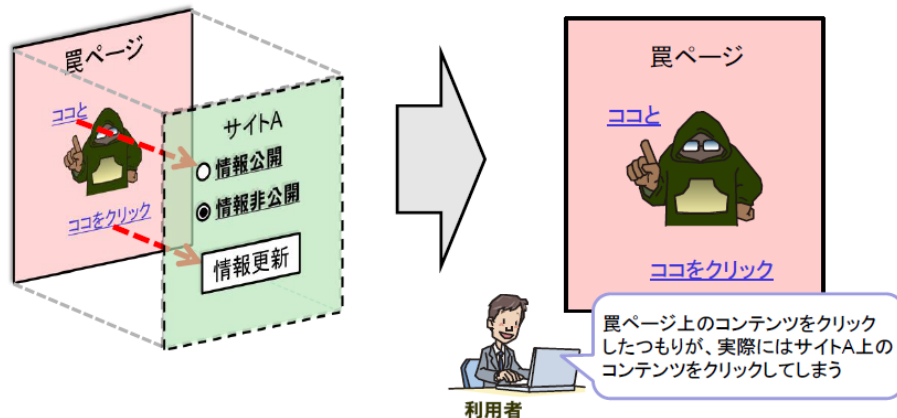
- ◆ メールフォームの宛先がHTML上に記載されていたり、入力欄に改行等が入力できる場合、意図しない宛先にメールを送信させられる可能性がある
- ◆ メール送信処理の際に、TOやBccといったヘッダ情報を固定値として処理する実装が必要となる。

11. クリックジャッキング

※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




◆ X-Frame-Options ヘッダの設定が行われていないページが、別のページに透過して重ねられることで、利用者が意図しない操作をさせられてしまう可能性がある

◆ X-Frame-Options:DENY の設定や、設定変更時のパスワード再入力等の対策が必要となる



ウェブサイトの運営に関わる**全ての人**に向けた内容

- ✓ “11種類の脆弱性”の説明とその対策を説明
- ✓ 運用面からのウェブサイト全体の安全性を向上させるための方策を説明
- ✓ 7版から“パスワードの運用方法”の内容を拡充
- ✓ ウェブセキュリティの対策状況を把握ができる**チェックリスト**つき

IPA 安全なウェブ 

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

参考：脆弱性体験学習ツール AppGoat IPA



脆弱性の概要や対策方法等の脆弱性に関する基礎的な知識を実習形式で学べるツール

どんなことが出来るの？



攻撃者視点

- 脆弱性の内容を学習
- 仮想環境に埋め込まれた脆弱性を**攻撃**

開発者視点

- 脆弱性の対策方法を学習
- 仮想環境に埋め込まれた脆弱性を**修正**

詳しくは

IPA AppGoat



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

第一章 見つけやすい11の脆弱性

第二章 フレームワークと共通部品に
必要な対策



- ウェブアプリケーションフレームワークが行うべきセキュリティ対策

1. 製品開発者として実施すべき対策

- 製品に脆弱性を作りこまないこと

2. 製品利用者が安全に利用するための対策

- ウェブサイトセキュリティのための機能
(エスケープ処理やセッション管理等)
を提供すること



1.製品に脆弱性を作りこまないこと(1)

- 製品の提供前に脆弱性検査を行うこと
 - 開発工程でのセキュリティ診断の実施
→意図しない動作が発生しないか確認
 - 脆弱性診断ツールを使用した検査
→診断の自動化、入力パターン・診断箇所の拡大
 - 専門家への相談
→製品の公開前にテストしてもらう等

1.製品に脆弱性を作りこまないこと(2)

- デバッグ用の機能についても検査が必要
 - デバッグ用の機能を実運用でも使用することや、デバッグモードで運用してしまい被害が生じる場合がある
→デバッグ用の関数や、デバッグモード時に有効になる機能に脆弱性が存在した
 - テスト用ページが存在する場合、実運用に入る際に削除が漏れてしまった例がある
→DBへのアクセスを確認するページ等、実際の運用には必要がないページの削除漏れ

例:

JVN#48237713

ADOdb におけるクロスサイトスクリプティングの脆弱性

1.製品に脆弱性を作りこまないこと(3)

- 脆弱性が発見された場合は速やかに修正等の対応と公表を実施すること
 - 特にフレームワークでは、利用者が独自に対応を行うことが困難
 - 速やかに修正バージョンを公開することが望ましい
 - 修正に時間がかかる場合は、ワークアラウンドを公開

フレームワークのアップデートには時間がかかる場合があり、アップデートまでの間に被害が生じる可能性がある

参考:Apach Struts2 の脆弱性被害

- ◆ 近年の傾向として、利用率が高い製品の脆弱性は極めて短期間に攻撃に悪用される

(日時は日本時間)

日付	時間	事象
2017年3月6日	19時ごろ	Apache より、S2-045に関する情報が公開される。
2017年3月7日	午前	中国のウェブサイトでPoCが公開される。
2017年3月7日	13時ごろ	中国国内で攻撃を検知。
2017年3月7日	17時ごろ	日本国内で攻撃を検知。
2017年3月7日	21時ごろ	Apache より、修正バージョンが公開される。
2017年3月8日	5時ごろ	保険特約料支払いサイトへの攻撃が発生。
2017年3月8日	11時ごろ	JPCERT/CCより早期警戒情報を公開。
2017年3月8日	14時ごろ	IPAより注意喚起情報を公開。
2017年3月8日	17時ごろ	都税支払いサイトへの攻撃が発生。
2017年3月8日	21時ごろ	Apache より修正バージョンのアナウンスメール送信。
2017年3月9日	午前	JPCERT/CCが注意喚起を公表。
2017年3月9日	18:00	GMO-PGがS2-045を把握。対象サイトの調査を開始。
2017年3月9日	20:00	GMO-PGにて対象となるシステムの洗い出しが完了。
2017年3月10日	0:30	GMO-PGにて保険特約料支払いサイトと都税支払いサイトへの不正アクセス発生を確認。

情報公開への注意と、迅速な修正版の公開が必要

2.ウェブサイトセキュリティのための機能を 提供すること(1)

- エスケープ処理やセッション管理等の機能を提供する

- エスケープ処理やセッション管理等は、重要機能であるため、標準の機能として提供される方が良い

過去には誤った対策として、エスケープ処理やセッション管理を
ウェブページのJava Scriptにより実装していた例がある



ブラウザの開発者ツール等で簡単に回避できてしまう

- ウェブアプリケーション開発者がエスケープ処理等の機能の必要性を理解していないことも原因

→機能や関数の存在と必要性について、マニュアル等で周知を行うことも
対策の一つ

2.ウェブサイトセキュリティのための機能を 提供すること(2)

- エスケープ処理はフレームワーク利用者がアプリケーション側でやるべきとの見解も
 - エスケープ処理の対策を、フレームワークが担当するか、ウェブアプリケーションが担当すべきか、明確な基準が存在しない
→しかし、どのような入力値の場合、異常な処理が発生するか、利用者が完全に把握することが出来ない
 - セキュリティ対策では、複数の対策により被害を防止することが重要
→単一の対策ではなく、複数の対策手段を実施出来ることが望ましい
例：ウェブアプリケーションでの入力値検査を回避された場合でも、フレームワーク側での出力値検査で遮断する

2.ウェブサイトセキュリティのための機能を 提供すること(3)

- デバッグ用関数等では、設計上脆弱性を排除できないことも考えられる

- 実運用よりも詳細なログを取得する等の理由でよりクリティカルな設計をする必要がある場合等

例: ファイルサイズの計測やメモリの解放などで
OSのコマンドラインの呼び出しが必要な関数

- マニュアルに危険性を記載し、実運用では使用しないよう注意喚起する

- デバッグモード等のまま運用することの危険性を周知する
- ウェブアプリケーション開発時にチェックすべき項目を公開する
→ 危険性が利用者に伝わらなければ、利用者は対処できない

2.ウェブサイトセキュリティのための機能を提供すること(4)

● 各フレームワークで実際に使用されているセキュリティ機能例

■ Apache Struts2

- SessionAwareインターフェース
 - セッション管理のためのStruts2標準の機能
- TokenInterceptorクラス
 - トランザクショントークンを検査するための標準の機能
- エスケープ処理はテンプレートエンジンやライブラリにて提供

■ Spring Framework

- Spring Security
 - 認証・認可やセッション管理機能を提供するフレームワーク
- エスケープ処理はテンプレートエンジンやPHP関数にて提供

2.ウェブサイトセキュリティのための機能を提供すること(5)

- 各フレームワークで実際に使用されているセキュリティ機能例

- Ruby on Rails

- session関数
 - セッション管理のためのRails標準の機能
 - form_authenticity_token関数
 - CSRFトークンの機能を提供するRails標準の機能
 - ページ上への表示ではRailsのviewアーキテクチャが自動的にエスケープ処理を行う
 - SQL等の入力はsanitize_sql_like関数等のエスケープ処理が提供されている

- 11の脆弱性について

- 脅威度が高い、SQLインジェクションやOSコマンド・インジェクション
- 発見されやすいクロスサイト・スクリプティング

入力値を直接ページに出力していることや、
関数への入力値を適切にエスケープしていないことが主な原因

- フレームワークに必要な対策

- 開発者: 脆弱性を作りこまない
- 利用者向け: セキュリティ対策のための機能を提供する

脆弱性が発見された場合は速やかな修正・ワークアラウンドの公開が必要

利用者がウェブフレームワークを安全に利用するためには、
開発者が利用者と目線を合わせて開発・提供することが必要です

ご清聴…
ありがとうございます
ございました!!

