



INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

PostgreSQLセミナー 2009春

～セキュリティ事故のケーススタディ～

独立行政法人 情報処理推進機構(IPA)
セキュリティセンター

相馬 基邦

対象・ポイント

- 対象

- 企業等のウェブサイト公開における安全性向上について理解を深めたい方

- ポイント

- 主に企業ウェブに関連したセキュリティ事故のケーススタディによる脅威と対策の技術的解説

本講の内容

1. SQL インジェクションとは？
2. 事例:ショッピングサイトでウイルス感染？
3. SQL インジェクションの対策
4. 安全なウェブ開発のための資料



1. SQL インジェクションとは？

1.1 SQLインジェクションとは

- 対象
 - データベースをバックエンドで利用しているウェブアプリケーション
- 原因
 - データベースへの命令の組み立て方に問題



1.1 SQLインジェクションとは(続き)

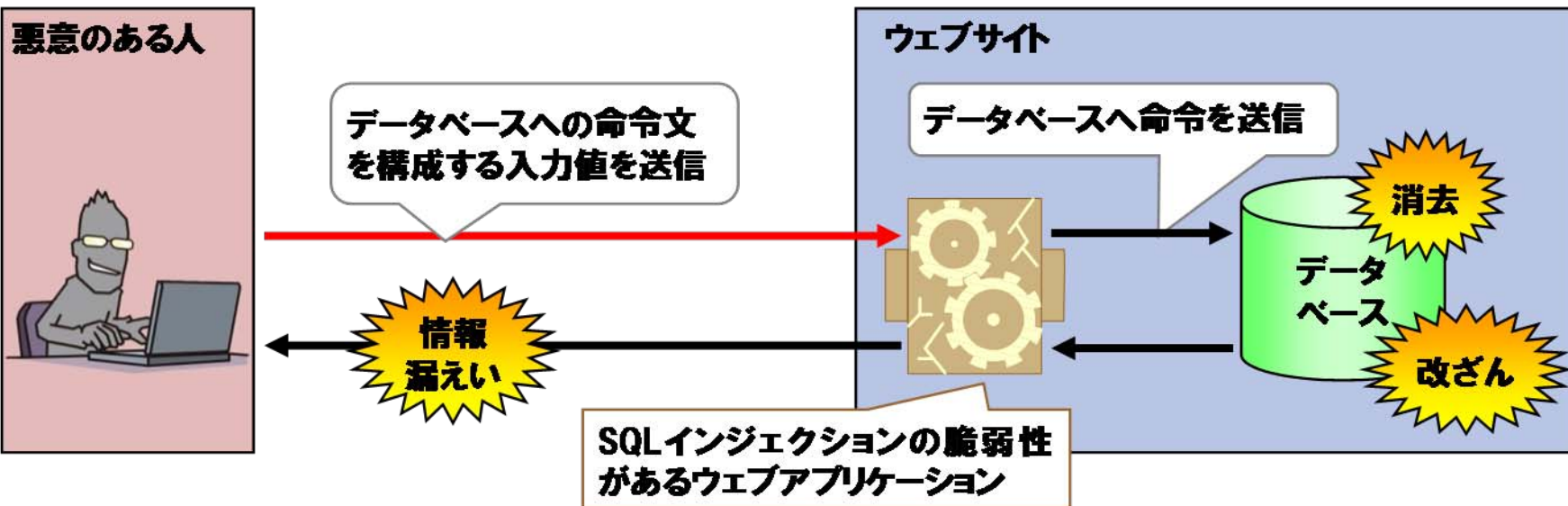
- 生じる脅威
 - データベースを直接操作されてしまう
 - ↓
 - 秘密情報、個人情報等の漏えい
 - 重要情報の改ざん、破壊
 - ウェブサイト上にウイルスを埋め込まれる
 - 任意のコード・コマンドを実行される
 - 別のサーバを攻撃する踏み台となる



SQLインジェクションとは

SQL インジェクション

SQL インジェクションの脆弱性がある場合、悪意あるリクエストにより、データベースの不正利用をまねく可能性があります。



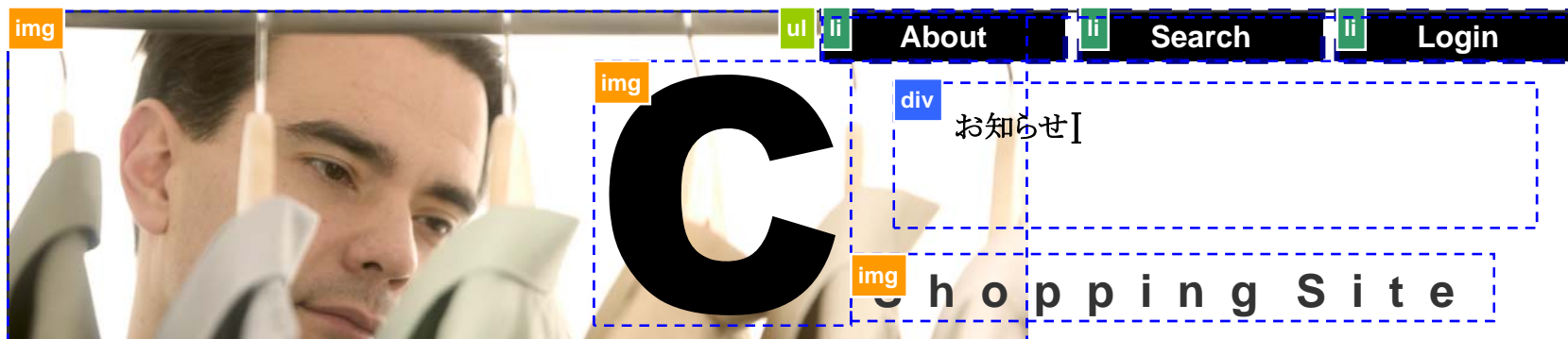
2.事例:ショッピングサイトでウイルス感染?

事例:ショッピングサイトでウイルス感染？



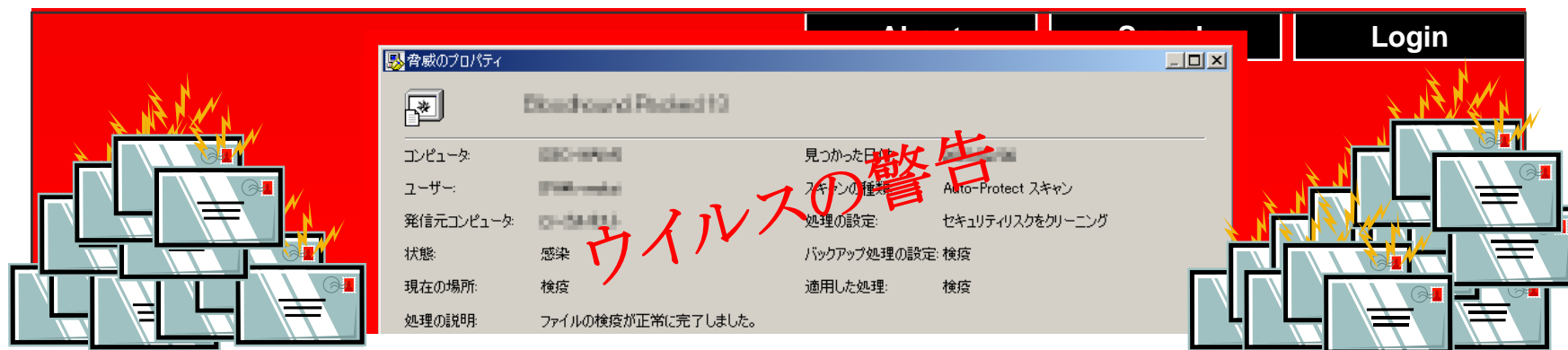
- ショッピングサイト C社。
- 社員数200名、Windowsのショッピングカートシステムを使った中規模ショッピングサイト。会員数10万人程度。
- リピーターがついており、アクセスは盛況。
- 最新パッチの適用、ユーザ管理、アクセス制限と、一通りは対策。

ショッピングサイトの裏側



- ショッピングカートシステムを自社開発
- ウェブのコンテンツはDBに格納されており、アクセスを受けるとショッピングカートシステムが動的にページを作成する。
- 販売担当者はショッピングカートシステムの管理画面から、自分の担当する商品情報を更新する。
- サーバ群の管理のために、管理者が数名。

ショッピングサイト上にウイルス



- ある日突然C社に、「ウェブを見たらウイルス対策ソフトが反応した」「ウェブサイトでウイルスに感染した」という苦情が複数寄せられる。
- ショッピングサイトにはユーザがページ内容を変更したり、ファイルをアップロードするような機能はない。
- 一体どこから？

現在の状態を調査する

- 外部に表示するウェブサイトをメンテナンス用サーバに切り替え、切り離れたサイトを調査。
- トップページに、悪意あるスクリプトを読みこませる内容を追加し、ウイルスに感染させようとしていた。
- 他のページには同様の埋めこみは発見できず。



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="ja">
<head>
<title>Welcome to "C" shopping site</title>
<link rel="stylesheet" href="style.css" TYPE="text/css">
<script src="http://othersite.example.net/3.js"></script>
</head>
```

外部のスクリプトを読みこんで実行

トップページに埋めこまれたHTML

ウイルスの正体は？

- 幸い、最新のアンチウイルスソフトを使うとウイルスを検知できた。
- 感染のために、ウェブブラウザの複数の脆弱性が使われている。ページを見るだけで感染する悪質なもの。
- 感染すると、マシンに潜み、キーロガー等の機能でパスワードやカード番号を盗みだす。
- アクセスログを見ると、数千人がダウンロードしてしまった可能性がある。



参考: 最近のウイルスの動作

- 罨ウェブ経由/普通のウェブ経由で感染
- ブラウザやプラグインの複数の脆弱性について侵入を試みる
- 最初は小さくて流用しやすいプログラム(ダウンローダ)に感染させる
- 実際の攻撃は別サイトからダウンロードしたコード(プログラムそのもの)が行う
- 何がダウンロードされるかわからない=対策パッチをあてても、そのパッチを掻い潜る次の攻撃手法が使われる
- ダウンロードや命令には、会社が制限できない内部→外部のHTTP/HTTPSを利用し、**正規の通信に見せかける**。

参考: 近年の標的型攻撃に関する調査研究

<http://www.ipa.go.jp/security/fy19/reports/sequential/index.html>

まずは感染経路を特定する

どうやってウェブサイトにウイルスが仕組まれたのか？

•想定1:

サーバへの攻撃？

サーバに侵入され改ざん

•想定2:

内部犯行？

•想定3:

ウェブサイト経由？

ショッピングカートシステムの脆弱性

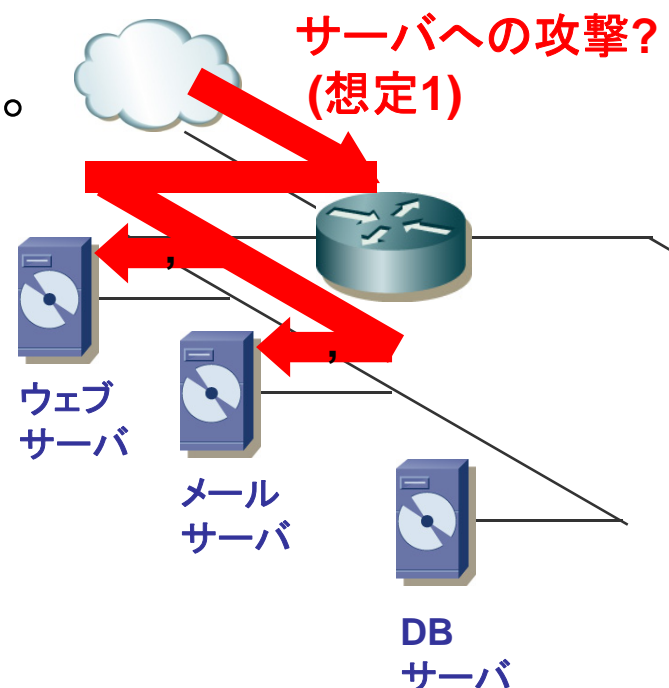


詳細な調査が必要

想定1: サーバへの攻撃?

外部からの攻撃による改ざんを疑う。

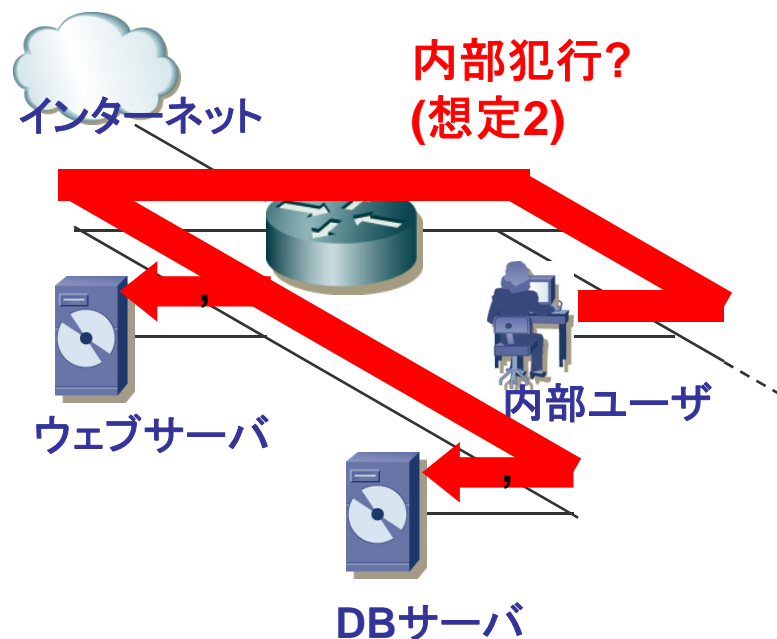
- 外部に公開しているのはウェブ用とメール用のサーバのみ。DBは非公開。
- 外部からはアクセス制限により、メール(port 25)と、ウェブ(port 80/443)しかアクセスできない。
- ソフトウェアは随時アップデートしており、既知の脆弱性の問題はない。
- 攻撃されたとすれば未知の脆弱性。他をまず調査すべき。



想定2：内部犯行？

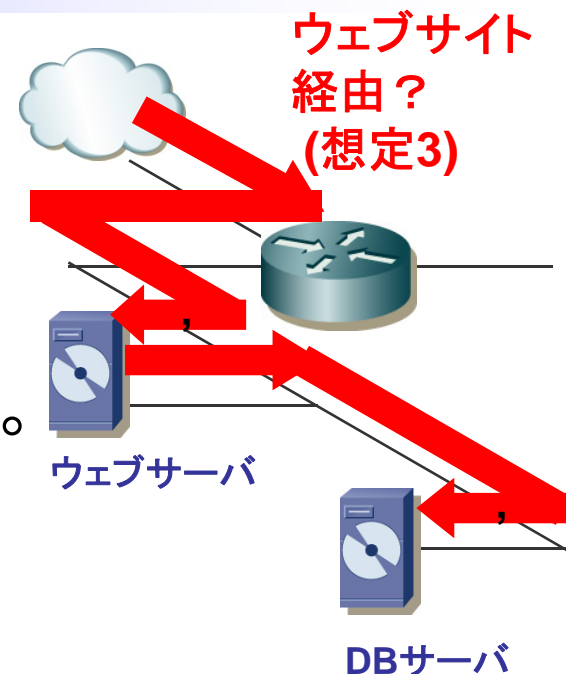
内部の人による書き換えを疑う。

- マシンに直接ログインできるのは管理者数人のみ。
- 商品の担当はショッピングカートシステムにログインして、担当個所を編集できるだけの権限しか持たない。トップページは無理。
- クロスチェックしたが、システムのログイン記録と、ショッピングカートシステムのログには、トップページの改ざんの記録や不自然なログの欠損は無かった。
- 内部からの可能性は薄そう。



想定3: ウェブサイト経由?

- ウェブサーバのログを見ていくと、エラーログの中に不審な文字列。
- 一見してSQL文のように見えるが、エラー。
- データベースにも同じようなログが...
- エラー表示は消しているので、SQL インジェクションではないはず...



```
/shop/cart/?no= ... %20SELECT%20 ... %20FROM%20 ... %20WHE  
RE%20 ... %20HAVING%20 ...  
(...)  
/shop/cart/?no= ... %20SELECT%20 ... %20FROM%20 ... %20WHE  
RE%20 ... %20ORDER%20 ...
```

ウェブサーバのエラーログ

SQLインジェクションの可能性あり

- 詳しく調べた所、データベースのログに、定型外の、データの内容を更新する SQL 文が記録されていた。

```
UPDATE pages SET "created_at" = '2006-09-05 16:19:46',  
"template" = 'toppage.tpl', "verified" = 1, "role" = NULL,  
"published" = 0, contents="<html><head> ...
```

データベースのログ

- これは、ショッピングカートシステムで使うデータベース内容を直接書換えるSQL文と判明。
- ショッピングカートシステムのログに残らない形でトップページを改ざん。悪意あるスクリプトを読みこませる内容を書きこんでいた。

ウェブの改ざんだけではなかった

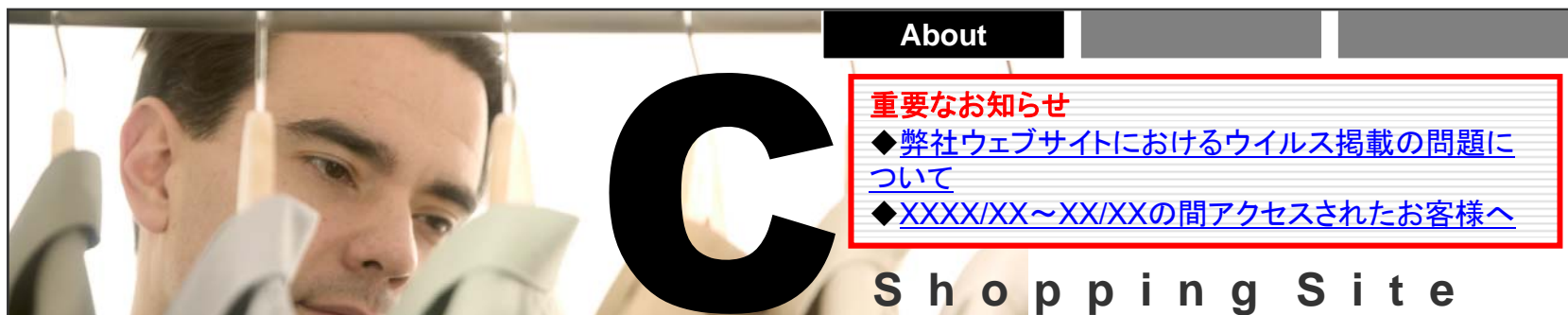
- データベースのログを更に確認。

```
... EXECUTE xp_cmdshell 'echo' || ( SELECT ... FROM ... ) || ...  
... EXECUTE xp_cmdshell 'echo' || ( SELECT ... FROM ... ) || ...
```

データベースのログ

- 順番通りに実行すると、データベースの内容を元に、その内容を OS のコマンドで送信するという内容。
- ウェブの改ざんの裏で、情報漏えいも起きた可能性があった。
- しかし、デフォルトで利用できない機能のため問題なし。

運営者としての対応

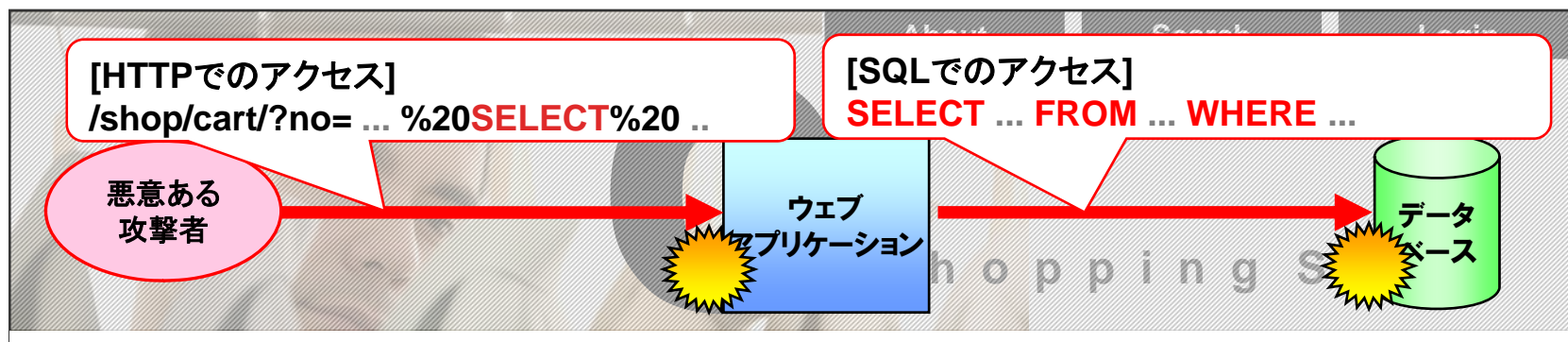


- 他にあやしいものが無いか、サーバやデータベースの内容を全面的にチェック。
- ショッピングカートシステムを使わずに、おわびと経緯の解説文章をウェブページに。駆除方法の提供。
- ショッピングカートシステムを修正。本来のウェブページは、**ショッピングカートシステムが修正された後**に公開。ウェブの営業停止による金銭的被害。
- 多層防御の一環として、ウェブアプリケーションファイアウォール(WAF)やサーバ用のアンチウイルス製品を導入。

参考:企業における情報セキュリティ事象被害額調査

<http://www.ipa.go.jp/security/fy18/reports/virus-survey/index.html>

何が起こっていたのか



- ウェブサイトに SQL インジェクションの脆弱性があり、そこを攻撃された。
- SQL インジェクションにより、ショッピングカートシステムのログに残らない形で、ウェブページを改ざんされた。
- さらに、OS コマンドが実行されて情報漏えいを起こされる可能性があった。

問題のポイント

- 自社開発ショッピングカートシステムに未知の脆弱性があり、自社ウェブサイト経由で、外部からのSQL文の挿入、およびデータベースの改ざんが可能であった。
- SQL文を構築する際のコーディング上の問題。

参考：脆弱性関連情報を発見してしまった場合は、脆弱性関連情報に関する届出について

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

3. SQL インジェクションの対策

SQLインジェクション対策のポイント

- ウェブアプリケーションで修正の必要がある。
- データベースの主要4機能(CRUD)を制御される。**脅威は情報漏えいに限らない。**
 - Create (作成) : 偽データの追加の脅威
 - Read (読込) : データの漏えいの脅威
 - Update (更新) : 偽データでの上書きの脅威
 - Delete (削除) : データの削除の脅威
- 他の脆弱性を利用するための前準備にも使われる。

SQL における「CRUD」

Create	(作成)	=	INSERT
Read	(読込)	=	SELECT
Update	(更新)	=	UPDATE
Delete	(削除)	=	DELETE

- 自由にこれらの命令が悪意を持ち利用された場合...
- データ定義の CREATE や ALTER、TRUNCATE、DROPなども利用される可能性

なぜ SQL 文の挿入が可能か？

- 特別な意味を持つ記号文字の扱いが不適切。
例：' (シングルクォート) : テキスト文字の引用符

```
SELECT * FROM a WHERE id=' $p' ;
```

\$p=foo の場合 :

```
SELECT * FROM a WHERE id=' foo' ;
```

\$p='foo' or 'a'='a' の場合 :

```
SELECT * FROM a WHERE id=' foo' or 'a'='a' ;
```

変数中の記号文字が意味のある文字として解釈される。

根本的解決と保険的対策

- 根本的解決
 - 「脆弱性の原因を作らない実装」を実現。
 - その脆弱性による攻撃を完全に無効化。
 - 可能な限り、根本的解決を行うのが望ましい。
- 保険的対策
 - 攻撃による影響を低減する「セーフティネット」。
 - 脆弱性の原因は依然として残る。
 - 保険的対策のみに頼る取組は望ましくない。

SQLインジェクション対策

- 根本的解決
 - エスケープ処理
 - バインド機構の利用
 - バインド機構以外でのエスケープ
 - エスケープ以前の問題
- 保険的対策
 - エラーメッセージの非表示
 - データベースアカウントの権限見直し
 - その他の対策

エスケープ処理(根本的解決)

- エスケープ処理の実施。

特別な意味を持つ記号文字が普通の文字として解釈されるように処理する。

例：' → '' (同じ文字の繰り返し)

\$p=foo' or 'a'='a の場合：

```
SELECT * FROM a WHERE id='foo'' or ''a''='a';
```

変数中の '(シングルクォート)が、普通の文字として解釈される。

エスケープ処理の方法

大きくわけて次の2種類。

- **バインド機構**の利用

(プレースホルダ、バインド変数、準備された文(Prepared Statement))

- **バインド機構以外でのエスケープ**

- エスケープ関数

(Perl の DBI quote() や PHP の dbx_escape_string())

- 置き換え演算子等で自己エスケープ処理

(s/'"/g; など)

エスケープ処理の実装例

- ・ バインド機構を利用（例: Perl DBI）
 - 独自の処理でエスケープ処理をする必要がなくなる。

```
$sth = $dbh->prepare(  
    "SELECT id, name, tel, address, mail FROM usr  
    WHERE uid=? AND passwd=?");  
$sth->execute($uid, $passwd);
```

バインド変数

プレースホルダ

参考: セキュアプログラミング講座

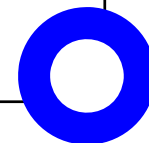
第6章 入力対策 SQL注入: #1 実装における対策

<http://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/502.html>

エスケープ処理の実装例

- エスケープ関数を利用(例: Perl DBI)
 - 可能ならば、バインド機構を推奨

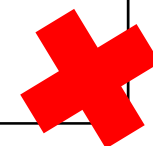
```
$sth = $dbh->prepare(  
    "SELECT id, name, tel, address, mail  
    FROM usr WHERE uid= "  
    $dbh->quote($uid) .  
    "AND passwd= "  
    $dbh->quote($passwd)");  
$sth->execute();
```



エスケープ処理の失敗例

SQL文の組み立てにパラメータ値をそのまま利用

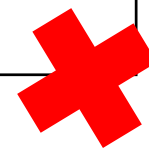
```
$sql="SELECT id, name, tel, address, mail FROM user  
WHERE uid=' ". $uid. "' and passwd=' ". $passwd. "'";  
$sth=$dbh->prepare("$sql");  
$sth->execute();
```



エスケープ処理の失敗例

もちろん、prepare 内で組み立てても駄目


```
$sth=$dbh->prepare(  
    "SELECT id, name, tel, address, mail FROM user  
    WHERE uid=' $uid' and passwd=' $passwd' ");  
$sth->execute();
```



エスケープ処理以前の問題(根本的解決)

- 外部から SQL 文を直接入力する。

```
<html>  
<form>  
  <input type="hidden" value="SELECT * FROM ...">  
</form>
```



- 「論外」であり、避けるべき実装であるが、実在する

エラーメッセージを表示すると・・・



■名前: 悪人 太郎 **SQL error with query SELECT a.name, a.displname, a.passwd, a.expire_date, a.create_date, a.misc FROM account as a WHERE a.category=7 ORDER BY c.date: Can't open file: 'account.MYD'. (errno: 145)**

■趣味: クラッキング

- 攻撃者の視点では、こう見える
 - データベースを利用→SQL インジェクションの可能性。
 - SQL エラーに気を使っていない→攻略の可能性。
 - エラー内容にSQL文が含まれる→テーブル定義が推測できる&試行錯誤すればどんどん調査できる可能性。
 - エラー内容から攻略方法を検討・・・。

エラーメッセージを非表示にする (保険的対策)

- ウェブアプリケーションで対応。
 - アプリケーションでエラーメッセージ表示処理を用意して、それを呼び出す。
- データベースの設定で対応。
 - 設定で変更。
- ウェブサーバの設定で対応。
 - 設定でエラー時の応答を設定。
- エラーを表示するとしても、内容は最小限に。
- エラーを消すだけでは解決しない。ツールでまとめてやられれば一緒。

エラーメッセージを非表示にする

- データベースでの設定例
 - PostgreSQL

(postgresql.conf で)

SILENT_MODE = on ;標準出力・標準エラー出力に一切のログを表示しない

SYSLOG = 2 ;ログを syslog だけに出力する。標準出力等にはしない

- 完全にエラーを表示させないようにする場合の例。
- エラーを表示しないだけでは駄目で、エラーの内容を選別して、ログに記録するようしておく必要がある。

エラーメッセージを非表示にする (保険的対策)

- ウェブサーバでの設定例
 - PHP 5

(php.iniなどで)

display_errors=Off : エラーをHTMLとして画面出力するか

display_startup_errors=Off ; PHPの初期動作時点で起きるエラーをHTMLとして画面出力するか

他、

error_reporting ; エラーの表示内容を設定

log_errors ; ログにエラー出力を行うか設定

- エラーを表示しないだけでは駄目で、エラーの内容を選別して、ログに記録するようにしておく必要がある。

データベースの権限は制限する (保険的対策)

- 「**権限全部入り**」のアカウントは**使わない**。
 - × postgres
 - × sa (System Administrator)
- データベース毎に専用のアカウントを作り、権限を制限して使う。
- 既存のテーブルを読み書きするだけなのに、テーブル操作や管理等の権限はいらない。
- 権限を必要最小限にすれば、防げる攻撃もある。

その他の対策

(保険的対策)

- 収集する情報を見直す
 - 必要最小限の情報しか格納しない
- DBに格納する情報を見直す
 - 内部で保持しておけば済む情報もあるのでは
- パスワードはそのまま保存しない
 - ハッシュ値を保存する

iLogScanner の活用

解析結果のレポート例

47件のSQLインジェクション攻撃を検出

攻撃成功は0件

攻撃成功の可能性は検出されなかった。

終了ステータス: 成功
 解析日時: 2008/04/07 14:22
 解析対象ファイル: access_log, access_log.2, access_log.5, access_log.6, access_log.7, access_log.8
 検出数 : 計 47件

検出対象脆弱性	攻撃があったと思われる件数	攻撃が成功した可能性の高い件数
SQLインジェクション	47	0

「SQLインジェクション」は、データベースと連携を行うWebアプリケーションにおいて、SQL文を改ざんされ、意図していないデータベース操作が実行されてしまう問題です。データベース操作を行うSQL文を、ユーザからの入力値を用いて構成している場合に、想定外の入力を行うことによりSQL文の書き換えを行います。この脆弱性が存在する場合、アプリケーションが使用するデータベースアカウントの権限で、データベース操作を実行することが可能になります。不正にデータベースが操作されることにより、情報の漏洩、不正な更新・削除によるデータベースの破壊などの影響が考えられます。実行可能な操作はアプリケーションが使用するデータベースアカウントの権限に依存するため、不必要に大きな権限が与えられている場合、不正なプログラムの埋め込みなど、さらに被害が広がる可能性があります。
http://www.ipa.go.jp/security/vuln/vuln_contents/sql.html

対象URL: <http://www.ipa.go.jp/security/vuln/websecurity.html>

解析結果のログ例

解析結果ログの見方
 # ログファイル名
 # [行番号] [脆弱性種別] [攻撃が成功した可能性が高い] [該当するアクセスログ]
 # ※ 各項目はタブ区切りになります
 # ※ 攻撃が成功した可能性が高い場合、「●」がつきます
 # 以下、解析結果ログ

行番号	脆弱性種別	攻撃が成功した可能性が高い	該当するアクセスログ
6891	SQLインジェクション		118.64 -- [02/Mar/2008:20
6892	SQLインジェクション		118.64 -- [02/Mar/2008:20
6893	SQLインジェクション		118.64 -- [02/Mar/2008:20
9020	SQLインジェクション		128.138 -- [03/Mar/2008:0
9033	SQLインジェクション		197.234 -- [03/Mar/2008:1
9035	SQLインジェクション		197.234 -- [03/Mar/2008:1
9036	SQLインジェクション		197.234 -- [03/Mar/2008:1
9048	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9049	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9050	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9051	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9052	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9053	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9059	SQLインジェクション		70.21 -- [03/Mar/2008:04
9060	SQLインジェクション		70.21 -- [03/Mar/2008:04
9061	SQLインジェクション		70.21 -- [03/Mar/2008:04
9070	SQLインジェクション		5.193.190 -- [03/Mar/2008
9071	SQLインジェクション		5.193.190 -- [03/Mar/2008
9072	SQLインジェクション		5.193.190 -- [03/Mar/2008
9083	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9094	SQLインジェクション		3.183 -- [03/Mar/2008:04:
9119	SQLインジェクション		70.21 -- [03/Mar/2008:04
9121	SQLインジェクション		5.193.190 -- [03/Mar/2008
9182	SQLインジェクション		197.234 -- [03/Mar/2008:1
9514	SQLインジェクション		5.193.190 -- [03/Mar/2008
9515	SQLインジェクション		5.193.190 -- [03/Mar/2008
9516	SQLインジェクション		5.193.190 -- [03/Mar/2008
9548	SQLインジェクション		5.193.190 -- [03/Mar/2008
9549	SQLインジェクション		5.193.190 -- [03/Mar/2008
75149	SQLインジェクション		5.193.190 -- [04/Mar/2008
75150	SQLインジェクション		5.193.190 -- [04/Mar/2008
75151	SQLインジェクション		5.193.190 -- [04/Mar/2008
76374	SQLインジェクション		165.64 -- [04/Mar/2008:16
76375	SQLインジェクション		165.64 -- [04/Mar/2008:16
76551	SQLインジェクション		165.64 -- [04/Mar/2008:17
76552	SQLインジェクション		165.64 -- [04/Mar/2008:17
77202	SQLインジェクション		165.64 -- [04/Mar/2008:17
77203	SQLインジェクション		165.64 -- [04/Mar/2008:17
77204	SQLインジェクション		165.64 -- [04/Mar/2008:17
77561	SQLインジェクション		165.64 -- [04/Mar/2008:17

攻撃元のIPアドレス

攻撃のあった時刻

ウェブサイトの脆弱性検出ツール iLogScanner

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

iLogScanner の利用例

- IPA で公開している OSS iPedia のアクセスログを解析
- 最近SQL インジェクションがますます急増

1.解析対象のウェブサイト

- IPA のOSS iPedia
(オープンソース情報データベース)

2.解析したログの期間

- 2008年1月～6月

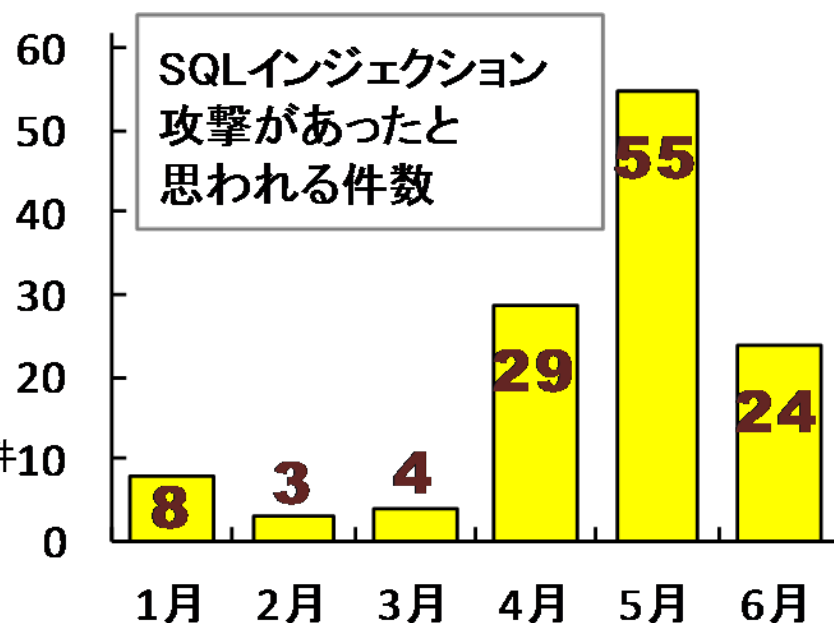
3.「iLogScanner」の解析結果

(1)攻撃があったと思われる件数:123件

(2)攻撃が成功した可能性の高い件数:0件

4.ログの詳細調査結果

- 攻撃に成功した件数:0件



まとめ:SQLインジェクションの対策

- SQL文の組み立てには必ずエスケープ処理を実装する。
 - バインド機構を推奨
- 公開ページでは、エラーメッセージをそのままブラウザに表示しない。
 - 表示しないだけでは駄目



安全なウェブ開発のための資料

他の脆弱性への対策も重要

- クロスサイト・スクリプティングやOS コマンドインジェクションに代表される脆弱性
- SQLインジェクションと同様に、設計時からの対策で防げるものが多い



- 実施する対策の効果や性質を正しく理解する
- 安全なプログラミング知識を身に付ける

安全なウェブサイトの作り方

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



書籍

ウェブ

- IPAに届出られた脆弱性関連情報をもとに、対策をまとめたもの
- 脆弱性ごとに解説と「根本的解決」「保険的対策」を記載
- 「失敗例」について記載
- ウェブセキュリティの実装状況のチェックリストつき