(ISC)² Secure Events

SECURE TOKYO 2017

Webサイトを安全にするための道具は揃っている

Tricorder Co. Ltd.
Sen UENO

Profile: 上野 宣 (Sen UENO)

株式会社トライコーダ 代表取締役

奈良先端科学技術大学院大学で情報セキュリティを専攻、eコマース開発ベンチャーで東証マザーズ上場などを経て、2006年にサイバーセキュリティ教育や演習、脆弱性診断、ペネトレーションテストなどを提供する株式会社トライコーダを設立

OWASP Japan 代表、情報セキュリティ専門誌『ScanNetSecurity』編集長、
Hardening Project 実行委員、JNSA ISOG-J WG1リーダー、SECCON 実行委員、
独立行政法人情報処理推進機構 セキュリティセンター研究員、
情報処理安全確保支援士 集合講習講師、セキュリティキャンプ講師主査 などを務める
(ISC)²が発表した2017年アジア・パシフィック情報セキュリティ・リーダーシップ・アチーブメント(ISLA)を受賞

主な著書

Webセキュリティ担当者のための脆弱性診断スタートガイド - 上野宣が教える情報漏えいを防ぐ技術、
HTTPの教科書、めんどうくさいWebセキュリティ、
今夜わかるシリーズ(TCP/IP, HTTP, メール) 他多数

Webサイトをセキュアにする7つ道具

安全にするための道具は揃っている

- Webサイトのセキュリティの問題は一通り出揃っている
- まったく新しいタイプのセキュリティの問題はそうそう登場しない

- OWASP Top 10
  - 最もクリティカルだと考えられるWebアプリケーションのセキュリティリスクのトップ10
  - 最新版は OWASP Top 10 - 2017 RC1 rejected
    - 2017正式版は近日リリース予定？
  - https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=Main

OWASP Top 10 - 2017 rc1

- A1： Injection(インジェクション)
- A2： Broken Authentication and Session Management(認証とセッション管理の不備)
- A3： Cross-Site Scripting(クロスサイトスクリプティング)
- A4： Broken Access Control(アクセス制御の不備)
- A5： Security Misconfiguration(セキュリティ設定のミス)

OWASP Top 10 - 2017 rc1

- A6： Sensitive Data Exposure(機密データの露出)
- A7： Insufficient Attack Protection(不十分な攻撃保護)
- A8： Cross-Site Request Forgery(クロスサイトリクエストフォージェリ)
- A9： Using Components with Known Vulnerabilities(既知の脆弱性を持つコンポーネントの使用)
- A10： Underprotected APIs(保護されていないAPI)

Webサイトをセキュアにする７つ道具

1. サイバーセキュリティ経営ガイドライン
2. OWASPアプリケーションセキュリティ検証標準（ASVS）
3. Webシステム／Webアプリケーションセキュリティ要件書
4. 安全なウェブサイトの作り方
5. Webアプリケーション脆弱性診断ガイドライン
6. Web Application Firewall（WAF）
7. グッド・プラクティス・ガイド パッチ管理

Webサイトをセキュアにする７つ道具

1. サイバーセキュリティ経営ガイドライン
   - サイバーセキュリティを経営課題とするために
2. OWASPアプリケーションセキュリティ検証標準（ASVS）
   - 自分たちに必要なセキュリティレベルを知るために
3. Webシステム／Webアプリケーションセキュリティ要件書
   - 安全なWebアプリケーションはどう設計すべきか
4. 安全なウェブサイトの作り方
   - 安全なWebアプリケーションはどう実装すべきか
5. Webアプリケーション脆弱性診断ガイドライン
   - Webアプリケーションをどうテストすべきか
6. Web Application Firewall（WAF）
   - Webアプリケーションへの攻撃の無効化、またはロギング
7. グッド・プラクティス・ガイド パッチ管理
   - 正しくパッチを適用してWebサイトを安全に保つ

1. サイバーセキュリティ経営ガイドライン

1. サイバーセキュリティ経営ガイドライン

- 経営者自らに**サイバーセキュリティは経営課題**であることを知ってもらい、それをCISOなどが実践するためのガイドライン

- サイバーセキュリティ経営ガイドライン(経済産業省)
  - http://www.meti.go.jp/policy/netsecurity/mng_guide.html
- サイバーセキュリティ経営ガイドライン解説書(IPA 独立行政法人 情報処理推進機構)
  - https://www.ipa.go.jp/security/economics/csmgl-kaisetsusho.html

サイバーセキュリティ経営3原則

① 経営者のリスク認識とリーダーシップが重要

② 自社以外（委託先やビジネスパートナーなど）にも配慮

③ 平時からの対策や情報開示など、関係者とのコミュニケーション・情報共有

重要10項目

1. サイバーセキュリティ対応方針の策定
2. リスク管理体制の構築
3. リスクの把握、目標と対応計画策定
4. PDCAサイクルの実施と対策の開示
5. 系列企業・ビジネスパートナーの対策実施及び状況把握
6. 予算確保・人材配置及び育成
7. ITシステム管理の外部委託
8. 情報収集と情報共有
9. 緊急時対応体制の整備と演習の実施
10. 被害発覚後の必要な情報の把握、開示体制の整備

経営者が責務を果たしているかどうかが判る問い

① ネットワークが1週間遮断された場合のビジネスへの影響度がわかりますか？
- 被害が生じた場合に、調査のためにネットワークを遮断する必要が生じることもある
- 優先するのはセキュリティ対策？事業継続？

② 自社への被害が想定されるサイバー攻撃について、社内に聞ける人がいますか？
- サイバー攻撃情報を収集できるCISOやCISOを補佐する役割として適任者の人材を知っておく

③ 漏えい事故や事件発生後にどのように状況を報告し、信頼を維持・回復しますか？
- サイバー攻撃発生時の説明責任は経営者にある

2. OWASPアプリケーションセキュリティ検証標準（ASVS）

2. OWASPアプリケーションセキュリティ検証標準（ASVS）

- 業界標準の**アプリケーション検証基準**
  - 設計・開発・テストに必要なセキュリティ要件および管理策のフレームワークの標準化
  - セキュアな開発ライフサイクル
  - 要求されるセキュリティレベルに応じて定義

- OWASPアプリケーションセキュリティ検証標準 (OWASP / 日本語訳 JPCERT/CC)
  - https://www.jpcert.or.jp/securecoding/materials-owaspasvs.html
  - https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

ASVSレベル

- レベル１ 導入（Opportunistic: 日和見的な）
  - すべてのソフトウェアを対象
  - 最低限満たすべきレベル
- レベル２ 標準（Standard）
  - 今日のソフトウェアが直面するほとんどのリスクに適応するアプリケーションが対象
  - 保護を必要とする機微なデータを含むアプリケーションを対象
- レベル３ 上級（Advanced）
  - 最高レベルの信頼性を必要とするアプリケーションを対象
  - 高い価値を伴う取引を行うものや機微な医療データを含むものなど

検証要件の詳細

1. アーキテクチャ、設計、脅威モデリング
2. 認証
3. セッション管理
4. アクセス制御
5. 悪性入力の処理
6. 暗号化
7. エラー処理とログの保存
8. データの保護
9. 通信
10. HTTPに関するセキュリティ設定
11. 悪性活動の管理
12. ビジネスロジック
13. ファイルとリソース
14. モバイル
15. Webサービス
16. 構成

例：V1: アーキテクチャ、設計、脅威モデリング [要件]

#	説明	1	2	3	導入
1.1	アプリケーションのすべての構成要素を把握し，それらが必要とされている	✓	✓	✓	1.0
1.2	ライブラリ，モジュール，外部システムなど，アプリケーションに内包されてはいないがその動作に必要な構成要素をすべて把握している		✓	✓	1.0
1.3	アプリケーションの高次のアーキテクチャが定義されている		✓	✓	1.0
1.4	アプリケーションのすべての構成要素が，業務上の機能，セキュリティ上の機能の観点で定義されている			✓	1.0
1.5	アプリケーションに内包されてはいないがその動作に必要となるすべての構成要素が，業務上の機能，セキュリティ上の機能の観点で定義されている			✓	1.0
1.6	対象となるアプリケーションの脅威モデルが作成されており，STRIDE，つまり，なりすまし (Spoofing)，改ざん (Tampering)，否認 (Repudiation)，情報漏洩 (Information Disclosure)，権限昇格 (Elevation of privilege) に関連するリスクが網羅されている			✓	1.0

3. Webシステム／Webアプリケーション セキュリティ要件書

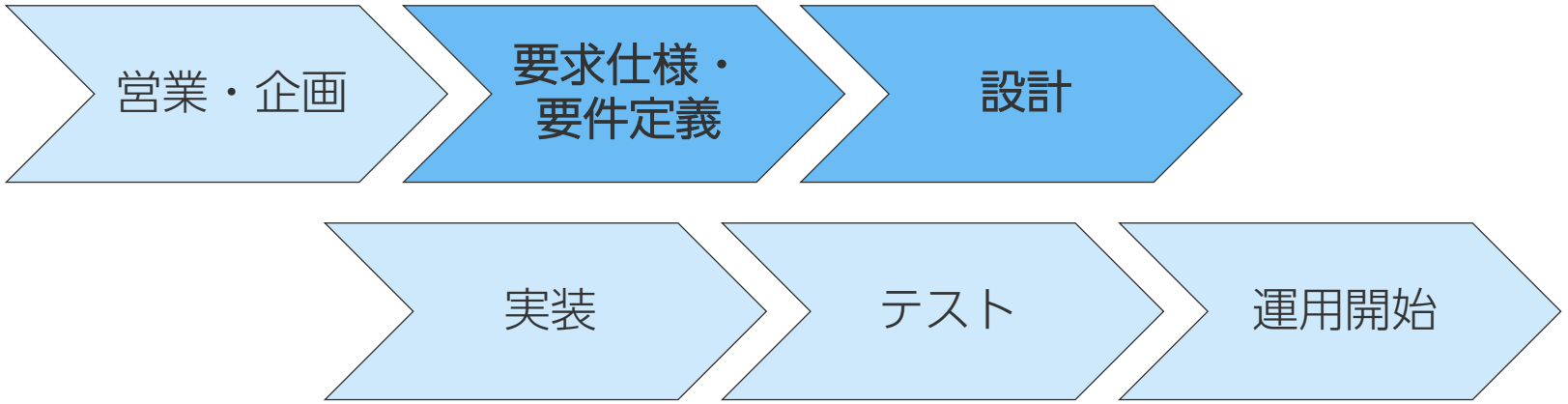
3. Webシステム／Webアプリケーションセキュリティ要件書

- 安全なWebアプリケーション開発の**具体的なセキュリティ要件書**
  - 安全なアプリケーションの開発
  - 開発会社と発注者の瑕疵担保契約の責任分解点を明確にする
  - 要求仕様やRFP（提案依頼書）として活用

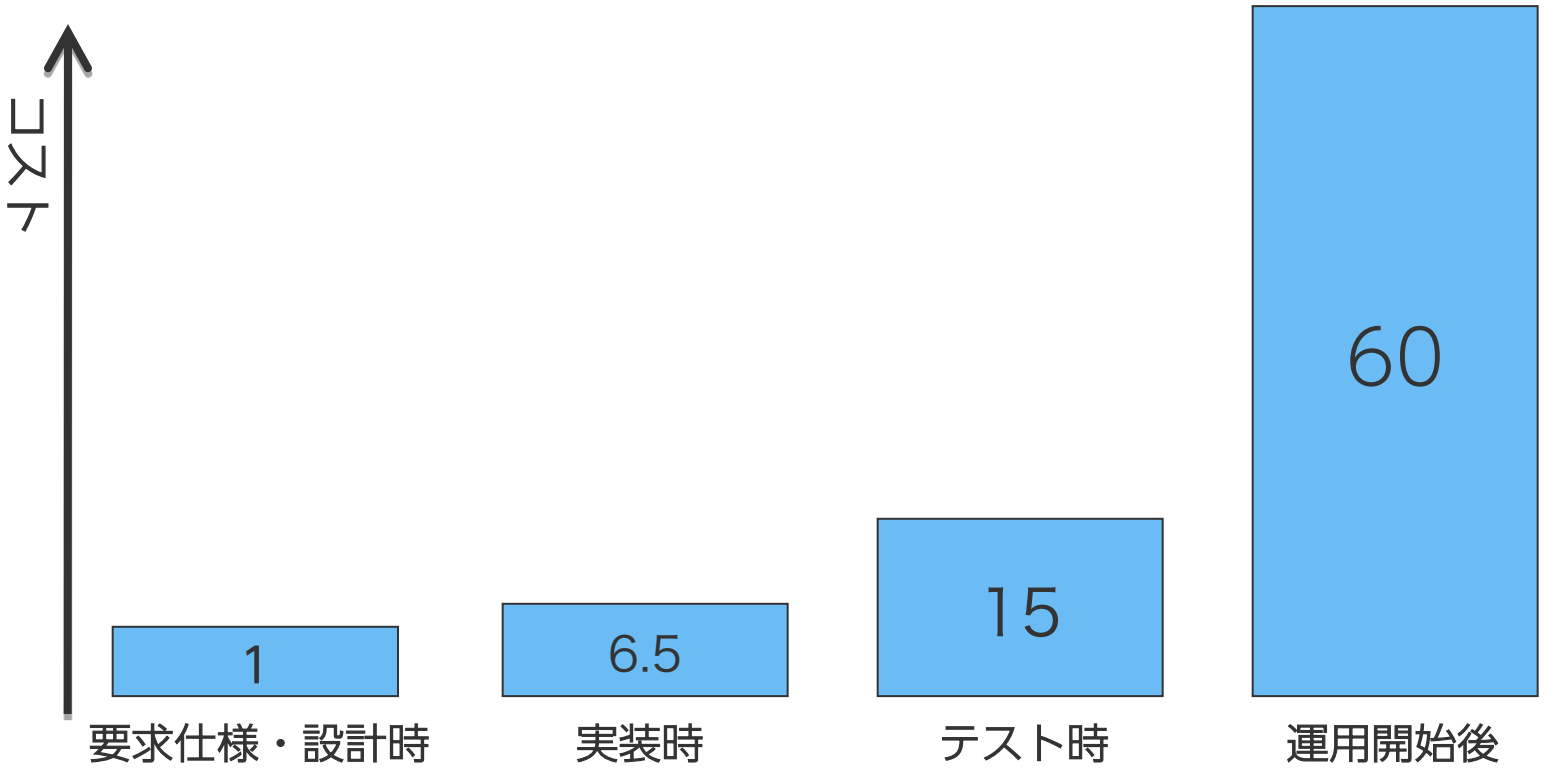
- Webアプリケーションセキュリティ要件書 (OWASP Japan)
  - https://github.com/ueno1000/secreq

安全なWebアプリケーションの開発には要件定義と設計が重要

- 安全なWebサイトを作るには設計以前が重要
- 早い段階で対応する方がコストが少ない
  - 要件定義や設計の不備は、後の全フェーズに影響



セキュリティホールを修正するコスト（フェーズ別）



参考：Kevin Soo Hoo, “Tangible ROI through Secure Software Engineering.”Security Business Quarterly, Vol.1, No.2, Fourth Quarter, 2001.

要求仕様項目

1. 認証
2. 認可（アクセス制御）
3. セッション管理
4. パラメーター
5. 出力処理
6. HTTPS
7. cookie
8. 画面設計
9. その他
10. 提出物

例：

3. セッション管理

3.1 セッションの破棄について

- 認証済みのセッションが一定時間以上アイドル状態にあるときはセッションタイムアウトとし、サーバー側でセッションを破棄しログアウトすること
- ログアウト機能を用意し、ログアウト実行時にはサーバー側でセッションを破棄すること
- ログアウト機能の実行後にその成否をユーザーが確認できること ※

認証を必要とする Web システムの多くは、認証状態の管理にセッション ID を使ったセッション管理を行います。認証済みの状態にあるセッションを不正に利用されないためには、使われなくなったセッションを破棄する必要があります。セッションタイムアウトの時間については、サービスの内容に応じて設定することが必要になります。

3.2 セッション ID について

- Web アプリケーション開発ツールが提供するセッション管理機能を使用すること
- 上記のセッション管理機能の使用が困難な場合、セッション ID は疑似乱数生成系により 80 ビット（使用する文字が 16 進数の場合 20 文字）以上の文字列を生成すること
- セッション ID をクライアントと受け渡しする際は cookie にのみ格納すること ※
- 認証に使用するセッション ID の発行は認証成功後とすること ※
- セッション管理機能を利用して機密情報を管理する場合、クライアントが送信した機密情報をサーバー側で受け取った時点でセッション ID を発行または再生成すること ※
- 認証済みユーザーの特定はセッション ID でのみ行うこと

セッション ID を用いて認証状態を管理する場合、セッション ID の盗聴や推測、攻撃者が指定したセッション ID を使わされるなどの攻撃から守る必要があります。フレームワークなどの制約によりセッション ID の発行を認証成功後に行うことが困難な場合には、ログイン後にセッション ID を再生成するなどの代替策が必要になります。
また、セッション ID は原則として cookie にのみ格納すべきです。

例：

5. 出力処理

5.1 HTML を生成する際の処理について

- HTML として特殊な意味を持つ記号（< > " ' &）を文字参照によりエスケープすること
  （ ' のエスケープはオプション扱い）
- 外部から入力した URL を出力するときは「http://」または「https://」で始まるもののみを許可すること
- <script>...</script>要素の内容やイベントハンドラ（onmouseover=””など）を動的に生成しないようにすること ※
- スタイルシートを外部サイトから取り込めないようにすること

外部からの入力により不正な HTML タグなどが挿入されてしまう可能性があります。「<」→「<」や「&」→「&」、「"」→「"」のようにエスケープを行う必要があります。スクリプトによりクライアント側で HTML を生成する場合も、同等の処理が必要です。実装の際にはこれらを自動的に実行するフレームワークやライブラリを使用することが望ましいでしょう。また、その他にもスクリプトの埋め込みの原因となるものを作らないようにする必要があります。
<script>...</script>要素の内容やイベントハンドラは原則として動的に生成しないようにすべきですが、jQuery などの Ajax ライブラリを使用する際はその限りではありません。ライブラリについては、アップデート状況などを調べて信頼できるものを選択するようにしましょう。

5.2 HTML タグの属性値を「"」で囲うこと

HTML タグ中の name="value"で記される値(value)にユーザーの入力値を使う場合、「"」で囲わない場合、不正な属性値を追加されてしまう可能性があります。

4.安全なウェブサイトの作り方

4.安全なウェブサイトの作り方

- セキュリティを初めて意識するWebアプリケーション**開発者向けドキュメント**
  - Webアプリケーションの脆弱性の理解と対策のための実装方法、失敗例
  - WebサーバーやDNSなどの対策

- 安全なウェブサイトの作り方 (IPA 独立行政法人 情報処理推進機構)
  - https://www.ipa.go.jp/security/vuln/websecurity.html

脆弱性対策について

- 根本的解決
  - 脆弱性を作り込まない実装
  - その脆弱性を狙った攻撃を無効化

- 保険的対策（セーフティネット）
  - 攻撃による影響を軽減する対策
  - 攻撃される可能性の軽減
  - 脆弱性を突かれる可能性の軽減
  - 被害範囲を最小化する
  - 被害が生じた場合に早期に知る

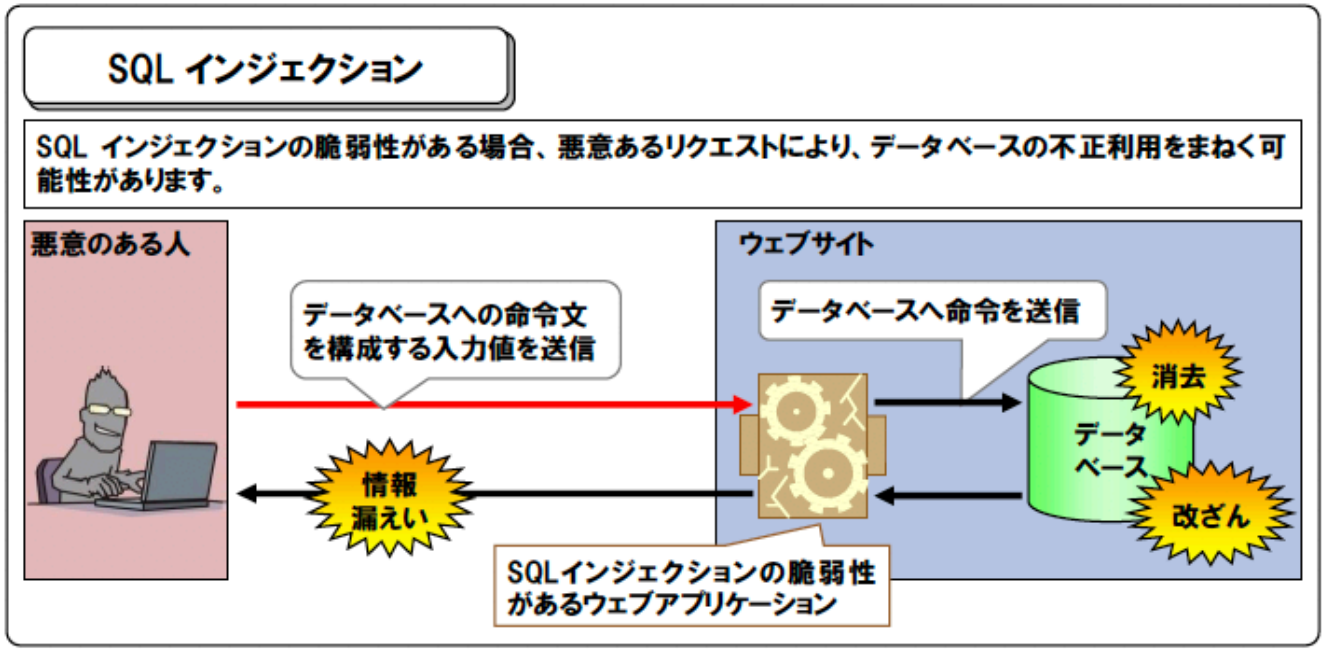
セキュリティ実装

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

例：

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。



例：

■ 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ります。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。

5. Webアプリケーション脆弱性診断ガイドライン

5. Webアプリケーション脆弱性診断ガイドライン

- 最低限必要なレベルのWebアプリケーション**脆弱性診断を行うためのガイドライン**

- 脆弱性診断士スキルマッププロジェクト(OWASP Japan)
  - https://www.owasp.org/index.php/Pentester_Skillmap_Project_JP

脆弱性とセキュリティ機能の不足

- 脆弱性とセキュリティ機能の不足を見つけるのが脆弱性診断
  - セキュリティテスト

- **脆弱性**
  - プログラムのバグ
  - バグの中で悪用可能なものが脆弱性
- **セキュリティ機能の不足**
  - 実装すればセキュリティレベルが向上する対策を実施していない
  - 安全でないプロトコルや弱いパスワードの利用など

Webアプリの脆弱性診断はゼロデイ探し

- Webアプリケーション（プロダクトやパッケージを除く）は**ゼロデイ脆弱性**を探すことになる
  - ゼロデイ脆弱性：公には周知されていない脆弱性
  - 攻撃者以外は誰も脆弱性を発見してくれない

プラットフォームで対策すべき事項

- **既知の脆弱性**
  - バージョンが古く脆弱性が存在する
  - すでにプロダクトのサポートが終了しているため脆弱性が修正されない
- **設定の不備**
  - 最新バージョンであってもセキュリティに関わる設定の不備による脆弱性が存在する

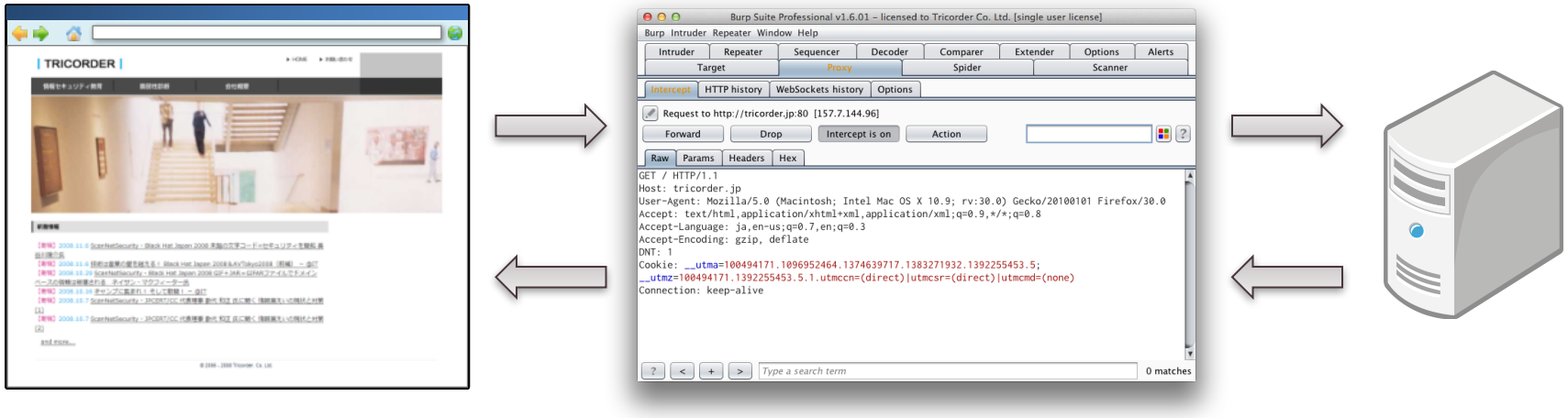
自動と手動の診断手法

- 自動的に脆弱性を発見する診断
  - 自動診断ツールによる診断
- 手作業で脆弱性を発見する診断
  - 手動診断補助ツールを使う手作業による診断

- 自動診断ツールでは発見が難しい脆弱性や、自動的に探すことが困難な機能や箇所があるため、確実に脆弱性診断を行うためには手作業による診断を合わせて行うことが必要
  - 自動診断ツールにしか発見できない脆弱性はない

脆弱性診断に使用するツール

- Proxyツール
  - WebブラウザとWebサーバーの間にProxyとして入り、リクエスト・レスポンスに介在する
  - 取得や保存、書き換え、再生などを行う



Webブラウザ　　Proxyツール　　Webサーバー

脆弱性診断に使用するツール

- 自動診断ツール
  - OWASP ZAP
    - OWASP：Webアプリケーションセキュリティのドキュメントやツールなどを作っているNPO
  - IBM Security AppScan, HP WebInspect, VEX

- 手動診断補助ツール
  - Burp Suite Free Edition
    - Professional版には自動診断ツールも搭載されている
  - Fiddler

Webアプリケーション脆弱性診断ガイドライン

- 手動診断補助ツールを使って脆弱性診断を行う際のガイドライン
  - 自動診断ツールによる脆弱性診断の検証にも活用

- ガイドラインの記載事項
  - 診断を実施すべき箇所
  - ペイロード・検出パターン
  - 操作を行う対象
  - 診断方法
  - 脆弱性がある／ない場合の結果

- 「ガイドライン利用のためのドキュメント」や拙著「脆弱性診断スタートガイド」と併用

例：

診断番号	診断対象の脆弱性	診断を実施すべき	検出パターン	診断を行う箇所	診断方法	脆弱性がある場合の結果	脆弱性がない場合の結果	備考
1	SQLインジェクション	すべて	「'」(シングルクォート1つ)	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	DB関連のエラーが表示されるか、正常動作と挙動が異なる	DB関連のエラーは表示されない	DB関連のエラー（SQL Syntax, SQLException, pg_exec, ORA-5桁数字, ODBC Driver
2	SQLインジェクション	すべて	(1)「(検索キー)」だけの場合と「(検索キー)' and 'a'='a」を比較／(2)「(検索キー)」だけの場合と「(検索キー)' and 'a'='b」を比較	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	(1)で同一の結果が返り(2)で異なる結果が返ってくる	(1)も(2)も同じ結果が返ってくる、または(1)も(2)も異なる結果が返ってくる	「' and 'a'='a」の部分がSQL文の一部として機能（演算を実施）している場合には、「'a'='a」は常に真（1）となり、判定結果に影響しないため、SQLインジェクションが可能であると判断できる ただし、この診断手法の脆弱性の有無については確定ではなく、あくまで可能性を示唆する
3	SQLインジェクション	型が数値のパラメーター	(1)「(検索キー:数値)」だけの場合と「(検索キー) and 1=1」を比較／(2)「(検索キー:数値)」だけの場合と「(検索キー) and 1=0」を比較	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	(1)で同一の結果が返り(2)で異なる結果が返ってくる	(1)も(2)も同じ結果が返ってくる、または(1)も(2)も異なる結果が返ってくる	「 and 1=1」の部分がSQL文の一部として機能（演算を実施）している場合には、「1=1」は常に真（1）となり、判定結果に影響しないため、SQLインジェクションが可能であると判断できる ただし、この診断手法の脆弱性の有無については確定ではな
4	SQLインジェクション	すべて	「||'」、「|||'」、「'+'」、「'++'」	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	文字列連結の演算が実行される	文字列としてそのまま評価される	DB処理が更新系の場合には実行しない方がいい場合もあるので注意
5	SQLインジェクション	すべて	1/0	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	演算が実行される（ゼロ除算のエラーになる）	文字列としてそのまま評価される	
6	コマンドインジェクション	すべて	||ping -nc 10 127.0.0.1%0a|	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	レスポンスが返ってくるのが20秒遅くなる	通常通りの応答速度でレスポンスが返ってくる	
7	コマンドインジェクション	すべて	../../../../../../bin/sleep 20|	パラメーター	パラメーターの値に検出パターンを挿入し、リクエストを送信	レスポンスが返ってくるのが20秒遅くなる	通常通りの応答速度でレスポンスが返ってくる	診断対象がUNIX系

6. Web Application Firewall （WAF）

6. Web Application Firewall（WAF）

- Webアプリケーション専用ファイアウォール
  - 不正な攻撃から守り、脆弱性を無害化
  - 攻撃を検知し、ログに記録

- Webアプリケーションにセキュリティ上の問題があっても無害化できる
  - 新たに発見された脆弱性に対して迅速に対応
  - 根本的なセキュリティ対策が不要なわけではない
  - 誤検知などの問題からログを取得する目的のみで運用することも

- 攻撃の痕跡を記録するためにも必要
  - 通常のWebサーバーが記録しているログだけでは、インシデント発生時に十分な調査ができない

Web Application Firewall（WAF）

- ソフトウェア型
  - Webサーバーにインストールするタイプ
  - ハードウェアなどの費用を抑えることができる
  - 設定や運用、障害の切り分けを行う必要がある
- アプライアンス型
  - ハードウェアとしてネットワーク内に導入するタイプ
  - 大容量の通信を処理できるものもある
  - 機密情報などを外部に出す必要がない
- クラウド型
  - クラウド上のWAFを利用
  - 設定や運用などを任せられることもある
  - CDNなどの機能と併せて提供されることもある

ModSecurity: Open Source Web Application Firewall

https://modsecurity.org/

7. グッド・プラクティス・ガイド
パッチ管理

7. グッド・プラクティス・ガイド パッチ管理

- すべてのシステムにパッチを正しく適用するためのガイドライン
  - パッチを正しく適用するための4段階の手順
- パッチを正しく適用すればプラットフォームの脆弱性は修正することができる
- グッド・プラクティス・ガイド パッチ管理 (CPNI / 日本語訳 JPCERT/CC)
  - https://www.jpcert.or.jp/ics/information02.html
  - 2006年と古いドキュメントだが基本は変わらない
    - 紹介されているツールは古い

4段階のパッチ管理プロセス

① 査定と棚卸し

- 運用環境がどのようなソフトウェアコンポーネントで構成されているか
- 新しいソフトウェア更新プログラムに対応するように準備できているか
- これらを正確に記録する

② パッチの識別

- リリースされたパッチを把握し、適合するかを判断、更新が通常なのか緊急なのかを判断する

③ 評価、計画、およびテスト

- 運用環境にパッチを展開するか決定し、実施計画、ステージング環境でのテストを実施し、問題がないかを確認

④ 展開

- 利用者への影響を最小限に抑えて運用環境にパッチを展開

OWASP Dependency Check

- Webアプリケーションに組み込んだコンポーネントのパッチ未適用を可視化するツール
  - アプリケーションが利用しているコンポーネントの種類、バージョンを調査し、最新版か否かを判断できる
  - Java, .NET に対応（実験的に Python, Ruby, PHP, Node.js, C/C++ にも対応）

- OWASP Dependency Check (OWASP)
  - https://www.owasp.org/index.php/OWASP_Dependency_Check

Vuls

- サーバー上の脆弱性を可視化するツール
  - システムに関係ある脆弱性のみ教えてくれる
  - その脆弱性に該当するサーバを教えてくれる
  - 自動スキャンのため脆弱性検知の漏れを防ぐことができる
  - CRONなどで定期実行、レポートすることで脆弱性の放置を防ぐことできる

- Vuls
  - https://github.com/future-architect/vuls/

時代に応じたセキュリティ対策が必要

過去のものになりつつあるもの

- パスワードの定期変更
  - 定期変更は漏えいしたパスワードが無効化できる程度で効果は限定的
  - 安全性が向上する認証機能や仕組みを提供すべき
  - NIST(米国国立標準技術研究所)のデジタル認証ガイドライン(SP800-63B)では「ユーザーにパスワード変更を要求すべきではない」とある (2017年6月)

過去のものになりつつあるもの

- SSL2.0は利用禁止, SSL3.0は廃止が求められている
  - SSL2.0（2011年3月利用禁止 RFC6176）
  - SSL3.0（2015年6月廃止を求める RFC7568）

- パスワード入力フォームをマスクする
  - スマートフォンの台頭で、入力のしづらさからマスクしない方が強固なパスワードを使用しやすいため、マスクを外すオプションが付けられることもある

完全HTTPS化

- 以前は、入力フォームや重要な箇所だけHTTPS化
- 多くのサービスが完全HTTPS化を進めている
  - Google, Facebook, Bing, Twitter, YouTube, Wikipedia, Netflix, Yahoo! JAPAN
  - HTTP/2はHTTPS利用が前提

- 完全HTTPS化のメリット
  - 盗聴・改ざん・なりすましの防止
  - Webサイトの信頼性向上
  - どのページをHTTPS化するかという議論不要
  - シンプルなセッション管理
  - HTTP/2を利用できるため通信速度が向上
  - Let's Encrypt などの無料の証明書もある

サイバー攻撃対策は"起こることを前提"に

- 以前は、攻撃を100%防ごうとしていた
- セキュリティ・インシデントは「起こることを前提」に考える
  - 完全にインシデントを防ぐことは不可能と知る
  - インシデントの防止は重要だが、インシデントの被害最小化が求められている

さいごに

- 必要なセキュリティ対策は時代によって変わることもある
  - 以前は効果があった対策が、意味がなくなることもある
  - 以前は推奨されていた対策が、今は推奨されない対策になることもある

- そのセキュリティ対策は今でも十分に機能していますか？
  - "7つ道具"を使って見直していきましょう

(ISC)² Secure Events

SECURETOKYO 2017