これからの
Webセキュリティを考える

～ ビジネスの生命線を守るには ～

住商エレクトロニクス株式会社
ネットワークセキュリティ事業部
技術担当副事業部長
二木真明

Webサイトはビジネスの生命線

- 単なる広告、宣伝媒体から直接的ビジネス窓口へ
  - コストをかけずに世界中から集客できる仮装店舗
  - 双方向のコミュニケーションによるマーケティング
  - 協業のための情報交換、共有、受発注効率化
  - 顧客の利便性の大幅な向上　　．．Etc．Etc.
- インタラクティブ性の向上とダイナミックコンテンツの増加
  - ECに利用する企業の増加
    - （約1／4の企業が利用　2004年版インターネット白書より）
  - 映像、音楽などの配布への利用
  - 急速なブロードバンド化の恩恵

2004/11/2

コンシューマーを相手にする業界では、すでにインターネットを使った顧客対応は必須といえる。インターネットの利点は言うまでもなく、ボーダレスかつ生活時間に影響を受けない（というよりは関係なく、自分の都合で使えてしまう）点である。この傾向は今後も増加するだろうし、B2B的なものにおいても、たとえばWebベースの業務連携などはどんどん進んでいくだろう。

ビジネスWebサイトのセキュリティとは

- **可用性 (Availability) の確保**
  - **ビジネスの生命線を「止めない」ことが至上課題**
  - **サービスの応答性 (処理性能) の維持**
- **完全性 (Integrity) の確保**
  - **提供する情報、受け取る情報の正確性の確保**
  - **意図的な情報改ざんや否認、証拠隠滅防止**
  - **システムの処理、動作の正確性の確保**
- **機密性 (Confidentiality) の確保**
  - **授受する情報の漏洩防止**
  - **システム内の情報の漏洩防止**
  - **システム構成情報の漏洩防止**

2004/11/2

セキュリティというと、CIAの視点で考えるべき・・・ということは、ISMSのようなものの浸透もあって、一般に言われ出した。この視点からWebサイトを見るとこのようになる。

特に、収益源をWebサーバにたよっているようなオンライン店舗主体の企業では、サイトの停止＝ビジネスの停止＝損失という図式がなりたち、どうしても、このようにA (可用性) がビジネス上の最優先課題になってくる。こうした優先順位は、たとえば「パッチをあてるためにサービスを一時的に停止する」とか、「パッチをあてたらシステムの動作が不安定になる可能性が否定できない」といったケースでの力学に大きな影響をあたえがちだ。一方、使う側から見ると、たとえば自分の個人情報がどのように管理されているかといった点が最も重要になるから、優先度は大きくかわってくる。個人情報保護法にみられるように、セキュリティのコンプライアンス化がどんどん進行すれば、Webサイト側も優先順位を再考せざるを得なくなるかもしれない。

Webサイトに対する外的な脅威

- ネットワーク経由の不正アクセス
  - システム、サービスへの攻撃
  - アプリケーションへの攻撃
- ネットワーク経由のサービス妨害
  - DoS, DDoS
- コンテンツの悪用
  - 著作物の不正利用
  - 掲示板、日記等を利用した迷惑、不正行為
  - 意図的なサイトの部分参照、引用
- 利用者の不正行為
  - なりすまし行為
  - 取引等の否認や意図的な歪曲
  - 詐欺的行為

2004/11/2

セキュリティを考える上で脅威の洗い出しは必須だ。Webサイト、ビジネスに対する脅威をざっと列挙するとこのようになるだろう。上の2つの項目はどちらかといえば、インフラに対する脅威、下の2つはコンテンツやビジネス自体に対する脅威と言える。

これまでのWebセキュリティ

- ファイアウォール、DMZによるサーバ保護
- サーバの管理徹底
  - 適切なアクセスコントロールの実施
  - 不要なサービス、アプリケーションの停止と削除
  - ログの取得、保全と定期的なチェック
  - セキュリティパッチの確実な適用
  - 不正プログラム対策の実施
  - セキュアなWebサーバソフトウエア設定
- 攻撃に対する監視と防御
  - IDS, IPSの導入と常時監視
  - ウイルス･ワーム対策の実施

どちらかといえば、コンテンツ（アプリケーション）を乗せる基盤のセキュリティを重視

2004/11/2

これまでのWebサーバのセキュリティは、どちらかといえばインフラの防御に重点をおいてきた。ここに書かれたレベルのことは、いまや主要なサイトでは、ほぼ実行に移されているものだ。（と信じたいのだが･･･）

狙われているコンテンツ

- 基盤への攻撃は難しくなってきた
  - 特にメジャーなサイトでは、従来型の対策は（一応）充実
  - 踏み台サイトは確保できるが、ターゲットを落とせない
- 興味本位の攻撃から目的のある攻撃へ
  - 攻撃、侵入が目的ではなく、手段になりつつある
  - 攻撃者の利益につながる情報の取得
  - 被害者の損害につながる情報の取得や妨害行為の実行
- 標的の変化
  - システム管理権限（root）奪取にこだわらず
  - より簡単に目的を達するには・・・
  - コンテンツやアプリケーションが新たな標的に

2004/11/2

インフラの防御が堅くなれば、当然、サイトに対して悪さを働こうという連中はやりにくくなる。どこかのサイトを攻撃する際に、自分の家のパソコンから直接攻撃する者がいるとすれば、よほどの素人か恐いもの知らずだろう。だいたい、どこかの管理が手薄でマイナーなサイトに侵入して、制御を奪ってから、本命を攻撃する。不幸にしてこうした踏み台にされてしまうようなサイトは残念ながら、まだまだたくさん存在するから、そこまではよいのだが、いざ標的を責めようとしても、なかなか攻め落とせない。

従来は、このような難攻不落と見えるサイトを、手間暇かけて攻略することにこだわる傾向もあったが、こうしたハッカーたちは、どちらかといえば技術的な興味で動いていた。要するに、そのサイトの中身よりも「落とす」ことが快感だったのだ。

しかし、一方で、「泥棒」たちにとっては、そのサイトをどう「落とす」かは問題ではなく、欲しいもの（情報）をどうやって得るかということが重要だ。「ハッカー」よりも「泥棒」が増えると、当然サイトに対する攻め方もかわってくる。「ハッカー」は root つまり管理権限を奪うことを最終の目標にすることが多いが、「泥棒」にとってはrootであろうが、一般権限であろうが、求めるものが得られればそれでいいのだ。従って「泥棒」はインフラではなく、守りが手薄なコンテンツを狙って攻撃することが多い。そして、「泥棒」に狙われたサイトのほうが、結果は深刻なものになる。

コンテンツへの脅威

- 非公開コンテンツへのアクセス
  - Webサーバ内にあるファイル等からの情報取得
  - 非公開のプログラム、スクリプトの不正実行
- Webアプリケーションへの攻撃や不正利用
  - 対話型アプリケーション利用における不正行為
    - 掲示板等の悪用（中傷、いやがらせ、プライバシー侵害、デマの流布・・・・・）
    - 取引等における詐欺、なりすましなどの行為
  - Webアプリケーションが持つ脆弱性の悪用
- 公開コンテンツの不正利用
  - 著作権侵害（不正使用、コピー・・・）
  - 悪意を持った引用、リンク
  - HTMLやスクリプト内に書かれた非表示情報の悪用

2004/11/2

Webサーバに置かれたコンテンツ（アプリケーションを含む）についての脅威を見てみよう。

たとえば、Webサーバのドキュメントディレクトリ下にあるファイルは、それがどこかにリンクされていなくても、直接ファイル名を指定することで、外部からアクセスができてしまう。たとえば、修正前のHTMLファイルやCGIスクリプトのバックアップファイル、極端な場合はCGIで使用するデータファイルなどがここに置かれていると、外部から盗み取られる危険が出てくる。ディレクトリが見えないからと言って安心できない。たとえば、拡張子が .BAK .ORG といったファイルがあるだろうことは容易に想像できるし、たとえば、user.dat 、password.dat などという名前も、「ハッカー」や「泥棒」にとっては想像しやすい。このようなファイルの中には、標的となる情報や、攻撃の参考になる情報が格納されていることもある。

Webアプリケーションの悪用、不正行為もまた脅威だ。掲示板などは書き込まれる内容によっては、設置者のリスクとなりかねない場合もあるし、ECサイトなのでは常に詐欺やなりすましといった危険にさらされている。さらに、最近クローズアップされているのが、Webベースのアプリケーションが持つ脆弱性だ。これらの脆弱性への攻撃は、インフラへの攻撃より一般に簡単なことが多い。ブラウザがあれば誰でも実行できてしまうものも多い。

公開されたコンテンツにも注意が必要だ。コンテンツを不正に利用される可能性がある。たとえば、本物のコンテンツを部分的に使用して、偽装したサイトを作る詐欺行為を行うといったことも考えられるだろう。また、よく、HTMLファイル内にコメントで様々な情報を埋め込んでいるサイトがあるが、これは場合によっては攻撃者に対して有用な情報を与えてしまう危険がある。ソース表示ですべて見えてしまう点に注意して余計な情報は書かない方が無難だ。また、スクリプト内に他のシステムをアクセスするためのパスワードなどは絶対に書き込んではいけない。すべて見えてしまうことになる。

非公開ファイルへのアクセス

- Webディレクトリ配下のファイルは直接アクセスが可能
  - 変更前のHTML、CGIスクリプトのバックアップ
  - アプリケーションが使用するデータファイル、データベース
  - 画像ファイル
  - アプリケーションのモジュール（サーバ側での実行）
- 推測可能なファイル
  - バックアップファイル（.bak .old .org …などの拡張子）
  - ソースファイル　（.c　……）
  - 特定の名前のファイルやディレクトリ（config, doc, man, ….）

ディレクトリは非公開でも、パスが推測できれば・・・・・・

2004/11/2

前のページで述べたとおり。

狙われるのはこのようなファイル。

非公開ファイルアクセスの危険性

- 管理用のデータの漏洩
  - 登録されたユーザIDやパスワードなどの情報
  - ユーザの個人情報
  - 各種のシステム情報
- 攻撃のヒント
  - 修正前の古いWebページに含まれる情報
  - HTML 内のコメントに含まれる情報
  - ソースコードに含まれたデータベースパスワードやファイル名、パス名など．．．．．

データファイルやバックアップ、ソースコードなどは少なくともWebサーバの管理外の領域（サーバ経由でアクセスできない領域）へ格納すること。

2004/11/2

非公開ファイルからは、おうおうにしてこれらのような情報が漏洩することがある。

少なくとも、サーバの文書ディレクトリの下には、公開ファイル以外は一切おいてはいけない。また、重要なファイルはWebサーバのアクセス可能範囲外に格納すべきだ。

Webアプリケーションの危うさ

- 経験があまりなくても簡単に作れてしまうWebアプリ
  - ルーズなプログラミングが可能で多機能なスクリプト言語
  - 便利な開発ツールの存在＝本当のプログラムの動きを知らないプログラマーの増加
  - 開発パワーの不足＝家内工業的開発委託先（二次、三次）の増加
  - 設計者の不慣れ＝設計レベルでの考慮不足
- ビジネスの急速なWeb化がもたらす危険
  - Webアプリと社内の重要システム（データ）とのリンク
  - セキュリティの面での検討不足
    - 社内システムとインターネットに公開されるシステムを結合するリスクが正しく評価されているのか？
    - 公開システムに対するセキュリティ要件は明確なのか？
    - 社内システム側の構成の見直しは不要なのか？

2004/11/2

Webアプリケーションの定義は広いが、ブラウザからアクセスして対話形式で操作が可能なサイトのように、Webサイト側でなんらかのプログラムが動いているようなものがほとんどだ。単純なperlスクリプトのようなものから、複雑なプログラミング言語で書かれた高機能なアプリケーションまで様々だが、開発に関していくつかの問題点が指摘できる。

まず、多くの場合こうしたアプリはスクリプト言語を使って記述される。最近のスクリプト言語は、変数型なども特に意識しないで使えるものが多く、便利ではあるが、一方でルーズなプログラミングをしても動いてしまうので、制作時にあまり熟考しなくなってしまうと恐い。プログラミング言語の容易さとロジックは別物だ。容易に書けるからと言ってロジック設計までルーズにしてしまうとバグ、すなわちセキュリティホールのもとになる点に留意すべきだろう。また、高機能の言語では、便利な開発ツールが完備しているものも多い。対話型で操作をしていくと、いつのまにかプログラムができあがってしまうので、プログラマーは言語の細部の動作を知らなくてもプログラムを開発できてしまう。実際、自分が開発ツール上で行った操作が、どのようなロジックに落ちるかを意識することは重要なのだが・・・。

また、Webアプリの流行で開発者も不足気味だ。二次、三次委託がすべて悪いとは言わないが、最終的に仕事をするプログラマーの能力までは掌握しきれないことも多いようだ。また、設計者も不足している。Webアプリは、セッション管理など独特の考え方が必要になることも多い。経験のない設計者はこうした部分をプログラマーに委ねてしまいがちだ。

また、Web化されたシステムをビジネスのフロントで使うためには、それらのシステムは既存の社内業務情報システムと接続される必要がある。インターネットに公開されたシステムを、もともと閉鎖的な発送で作られたシステムと接続することに問題はないのか・・・、といった点については十分に検討されなくてはいけないだろう。場合によっては、既存システム側を改修して安全性を高める、といった対応が必要になる可能性が高い。

Webアプリの注意点

- **認証が必要なアプリケーションのログイン管理**
  - HTTP(S)では、ログイン（セッション）管理はアプリケーションに実装しなければいけない。
  - 脆弱な実装を狙った攻撃により、アプリケーションの不正使用を許してしまう可能性がある。
- **不正な入力値排除の必要性**
  - 入力値がHTMLを構成するために利用される場合、HTMLタグを入力に混入することで、ブラウザの動作に影響を与えられる。（スクリプトの実行など）
  - 入力値を使用して他のシステムに渡すスクリプトやコマンドを生成する場合、特定の記号や予約語などの混入で、これらに影響を与えられる。（スクリプトの改変や不正なコマンドの実行など）
  - 入力値を使用してファイルアクセスのためのパス名を組み立てるような場合、意図しないファイルをアクセスさせられてしまうような可能性もある。
  - URL上のパラメータは常にユーザに見える上、ブラウザ上で簡単に改ざんできる。
  - フォーム内に固定で格納されている値は、クライアント側で改ざんできる。（Hidden パラメータ、選択肢が持つ固定値など、POSTでも改ざん可）
  - Cookie もユーザ側で改ざん可能

2004/11/2

Webアプリでよく問題とされる点を見てみよう。

たとえば、あるユーザがログインしていくつかの操作をし、ログアウトするまでの流れを「セッション」と呼ぶ。一つのセッションに属する操作は、その結果を次に操作に引き継ぐ必要があるし、当然前の結果が反映されていなければいけない。ところが、HTTP(S)における通信は、1ページ単位で完結してしまうため、Webサーバはアプリケーションを1ページのアクセスのごとに起動し、終了させる。従って、ログインから、ログアウトまでの間にユーザが何をしたかはアプリケーションが管理して適切な動きをしなければいけない。これをセッション管理と呼ぶが、アプリケーションが独自に実装しなければいけない。認証やユーザ管理とも密接にからむため、この部分に問題があると、アプリケーションの安全性も大きく損なわれる可能性がある。

また、ユーザはフォームに入力を行い「送信」ボタンを押すことで情報をアプリケーションに引き渡すが、アプリケーションが引き渡された値をどう使うかがわかっていると、意図的に入力に不正な値を混入することで、アプリケーションの動作に影響を与えられる可能性もある。たとえば、入力値をそのまま使って、送り返すページのHTMLを生成するようなケースや、入力値でデータベース問い合わせや、他のシステムへの要求、コマンドなどを生成するような場合は、こうした危険性が高い。

フォームには表に表示されない HIDDEN と呼ばれるフィールドが隠されていることがある。このフィールドの内容は、前の操作を受け付けたアプリケーションによってセットされ、ユーザがフォームを送信するとアプリケーションに送り返されるので、1回単位で完結してしまう操作の間でデータを受け渡すために使用されることが多い。Cookieと呼ばれる値は、あるサーバに固有の値として、アプリケーションからブラウザに渡され、ブラウザは以後、そのサーバにアクセスするたびにそのCookieを送り返す。アプリケーション側はセッションごとに含まれる値を一意にすることで、ある送信がどのセッションに属するかを認識できる。HiddenフィールドやCookieはふつうにブラウザを使用していれば、先に渡した値がそのままアプリケーションに戻されるが、悪意を持って操作すれば、その値を書き換えて送り返すことも可能だ。中身を見ることも当然可能なので、改ざんされてもいいような前提でアプリケーションを設計する必要がある。このような固定値に見られては困る値を平文で入れたり、返された値を無条件で信用して処理することは絶対に避けねばならない。

Webアプリの認証とセッション管理

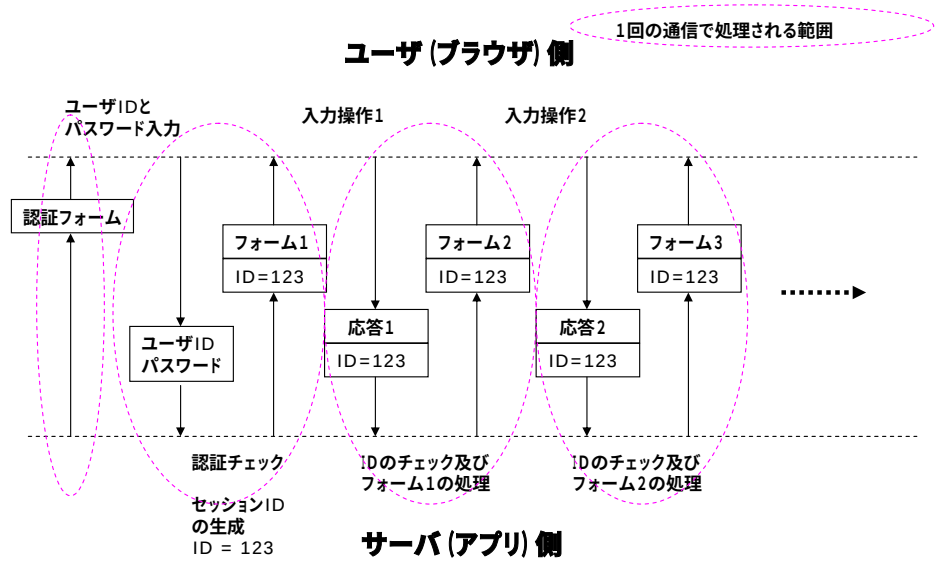
- HTTP(S) の特性
  - 1アイテム(HTTP1.0)または1ページを構成する程度の数のアイテム(HTTP1.1)を転送するような単位でしか、TCPのコネクションは維持されない。
  - 簡単な認証機構は存在するが、ログインからログアウトまで、複数ページにわたる対話型の接続を管理するための機構がない。
- HTTP(S)を使用したセッション管理
  - ログイン～ログアウトまでのユーザ管理はアプリケーションの責任
  - ログイン後、ユーザがたどるすべてのページアクセスで認証情報を引き継ぐ必要がある。

2004/11/2

セッション管理の細部を見てみる

HTTPについては、簡単な認証（ベーシック認証）機構はサポートしてはいるが、複雑なアプリケーションの認証に使用できるほどのものではない。従って、ログインからログアウトまでのユーザの操作に関する管理はすべてアプリケーション側で実装しなければいけない。

Webアプリにおけるセッション管理の例



2004/11/2

セッションの流れはこの図のようになる。ログイン時にその新たなセッションに対し一意的なIDを割り当て、なんらかの手段で、それ以降の操作が送信された際に、その値もアプリケーションに戻されるようにすれば、セッションを管理できる。

セッションID（識別情報）の受け渡し

- サーバから渡したセッションIDは、次の送信で送り返してもらう必要がある。
- URL引数を使用する方法
  - 送り返したページ内のリンクに組み込むことで、クリック時にURL の引数として受け取り
- フォームを使用する方法（Hidden Parameter）
  - フォーム内の隠しパラメータとして格納しておく方法
  - フォーム送信時に自動的に送り返し
- Cookie を使用する方法
  - Cookie 内のデータとして送信。
  - ブラウザは自動的にそのサイトに対してCookieを送信する。

2004/11/2

セッションIDの受け渡しの実装方法は、主にこれらの方法だ。最も良く使用されるのが、Cookieを使用する方法である。

セッションID受け渡しの留意点

- いずれの方法をとっても、ユーザ側でその内容を見たり改ざんできてしまう。
  - ユーザやパスワードなどが含まれたり推測できるような内容は、偽造を誘発する。
  - 数字などの場合でも、他の数字を入れてみるなどの行為が行われる可能性がある。
  - 同じ値を再利用される可能性がある。
- 情報の秘匿や改ざん、再利用のチェックが可能な対策が必要。
  - 内容の暗号化、時刻やシーケンス情報の付加など
  - SSLは経路途中の盗聴、改ざん対策であり、ユーザ側での行為に対しては無力。(必要だが不十分)

2004/11/2

セッションIDの受け渡し方法はいくつかあるが、どの方法を使用しても、セッション情報の秘匿、改ざんからの保護は難しく、それを前提としてチェックを行うような設計が必要になってしまう。Cookieは通常、HTTP(S)通信のヘッダに含まれて受け渡されるため、Hidden パラメータやURLのように表面に出ることはないが、ブラウザによってPCのディスク内に保存されることもあるので注意が必要。メモリ上のみで生存するクッキーも可能だが、それにしても特殊な攻撃ツールを使えば横取りや書き換えも可能な点に注意が必要。

従って、Cookieを含むセッション情報には、盗まれて困るような内容は決して入れないことが必要。どうしても入れる場合は、適切な強度で暗号化するなどして保護すること。また、戻された値は常に改ざんの可能性があるため、書き換えられていないことをチェックするための機構を検討すべきだ。

SSLを使えば安全との誤解もあるが、SSLは単に通信経路上での横取りや相手方詐称を防ぐ効果しかなく、パラメータ改ざんなどに対して有効な保護策にはなりえないので注意すること。(必要なのだが、十分ではない)

不正な入力値による攻撃の例

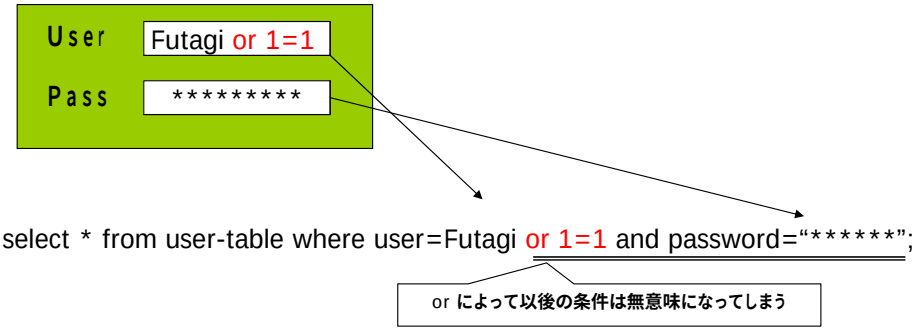
- SQL インジェクション
  - 入力値を使ってデータベースの検索、更新などを行うアプリが標的
  - 入力値に対し意図的にSQL構文を混入し、検索条件などを攪乱することが可能
- クロスサイトスクリプティング（ＸＳＳ）
  - 入力値を組み込んでページ（HTML）を生成するようなアプリケーションが標的
  - 入力値に対し、意図的に＜Script＞タグを混入して、出力されたページに他のサイトから不正なプログラムを読み込むなどの処理を組み込んでしまうことが可能

2004/11/2

不正な入力値によってアプリケーションに製作者が意図しない動きをさせるような攻撃の代表格を2つ紹介する。

SQLインジェクションの例

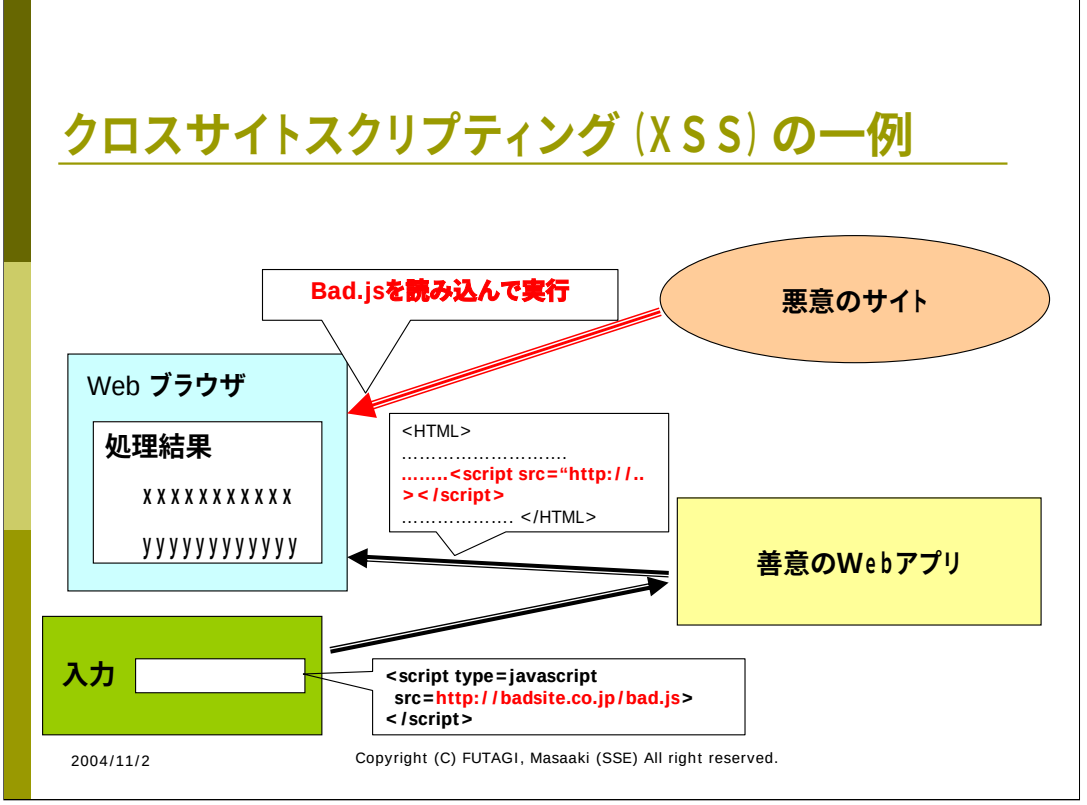
生成する検索条件 select * from user-table where user=User and password=Pass;



＊結果として、パスワードに関係なく問い合わせが成功してしまう。

きわめて単純なSQLインジェクションの例（あまり実際的な例は紹介できないので・・・）

たとえば、入力値から直接、問い合わせ分を作るような処理を行っていると、不正な値を入力することで、問い合わせの条件を操作できてしまうという例。



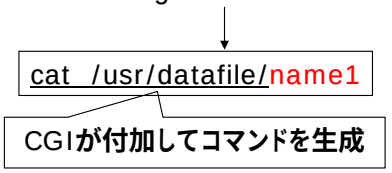
これも、単純な例。実際のXSSを詐欺的な用途に使うには踏み台サイトを作るなど、いくつかの手順が必要だが、不可能ではない。

これは、リターンページにスクリプトが埋め込まれてしまうことのインパクトを説明するための例。

その他の例

- CGIを介した不正なファイルアクセス

http://badsite.2bad/showfile.cgi? name1



http://badsite.2bad/showfile.cgi? ../../etc/passwd

なにが起きるだろうか・・・・・・・・

たとえば、OSのコマンドを生成するような場合にも問題は発生する。この例では、CGIを呼び出して、本来Webサーバ経由ではアクセスできないファイルを取り出してしまうことができるようなケース。

Webアプリにおける入力値処理

- 確実な入力値のチェックと利用を
  - 処理結果に影響をあたえるような記号、キーワードなどの排除またはエスケープ
  - 入力桁数のチェック
  - Hiddenフィールドや選択肢、スクリプト内の固定値もチェックの対象に（外部から与えられるすべての値には改ざんの可能性あり）

相手は正式な入力フォームとは限らない。フォーム上での桁数規制、スクリプトによるクライアント側での入力チェックは一切信用するな。

2004/11/2

もう一度、入力値チェックについて。

入力されることで、内部処理に混乱を引き起こすような文字、キーワードはすべて排除すること。

Hidden, URLパラメータ、Cookieなどは必ず、見られて、しかも、改ざんされうるという前提で設計すべき。

相手型はブラウザとも、その上で自分のアプリが先に送ったフォームとも限らない。従って、フォームで桁数規制が行われていても、スクリプトで入力値のチェックが行われていたとしても、それらを一切信用してはいけない。

Webアプリの安全性確保

- 開発者のレベルアップ（セキュリティ感覚の向上）
  - 脅威の認識（攻撃手法、事例など）
  - 仕様としてのセキュリティ組み込み
  - セキュアな開発手法の修得
  - 安全性確保は品質管理の一環であるという意識
  - 集団的な開発レビュー体制の確立
- 運用側のレベルアップ
  - 開発者任せにしない安全管理（必要に応じた検査・監査）
  - 日常運用おける危険察知と回避、対応策の準備

2004/11/2

Weアプリの安全性確保は、第一義的には開発上の問題。品質管理の一貫としてとり組みを進めるべき。一方、そのようなアプリを導入し、運用する側でも開発者任せにせず、チェックの体制、問題点の運用上での回避策などを考えておく必要がある。

開発者のための参考資料

- IPA (情報処理推進機構) HP
  - http://www.ipa.go.jp/security/awareness/vendor/software.html
- OWASP Guide to Building Secure Web Applications
  - http://www.owasp.org/documentation/guide/guide_downloads.html
- 安全なWebアプリ開発40箇条の鉄則 (高木浩光氏)
  - http://staff.aist.go.jp/takagi.hiromitsu/#2003.6.19
  - http://java-house.jp/~takagi/paper/idg-jwd2003-takagi-dist.pdf

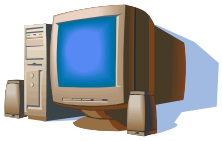
2004/11/2

IPAの資料は、非常によく整理されている。

高木氏の40箇条は、とてもわかりやすい。(べからず集的な色彩)

運用サイドから見た対策

- アプリケーションの脆弱性検査
  - 導入済みのアプリケーションの検査
  - 新規アプリケーション稼働前の検査
  - アプリケーション修正･追加後の検査
- 包括的な保護システムの構築
  - 攻撃に対する監視体制の構築
  - アプリケーションファイアウォールの利用



2004/11/2

運用側としての対応にはどのようなものが考えられるだろうか。

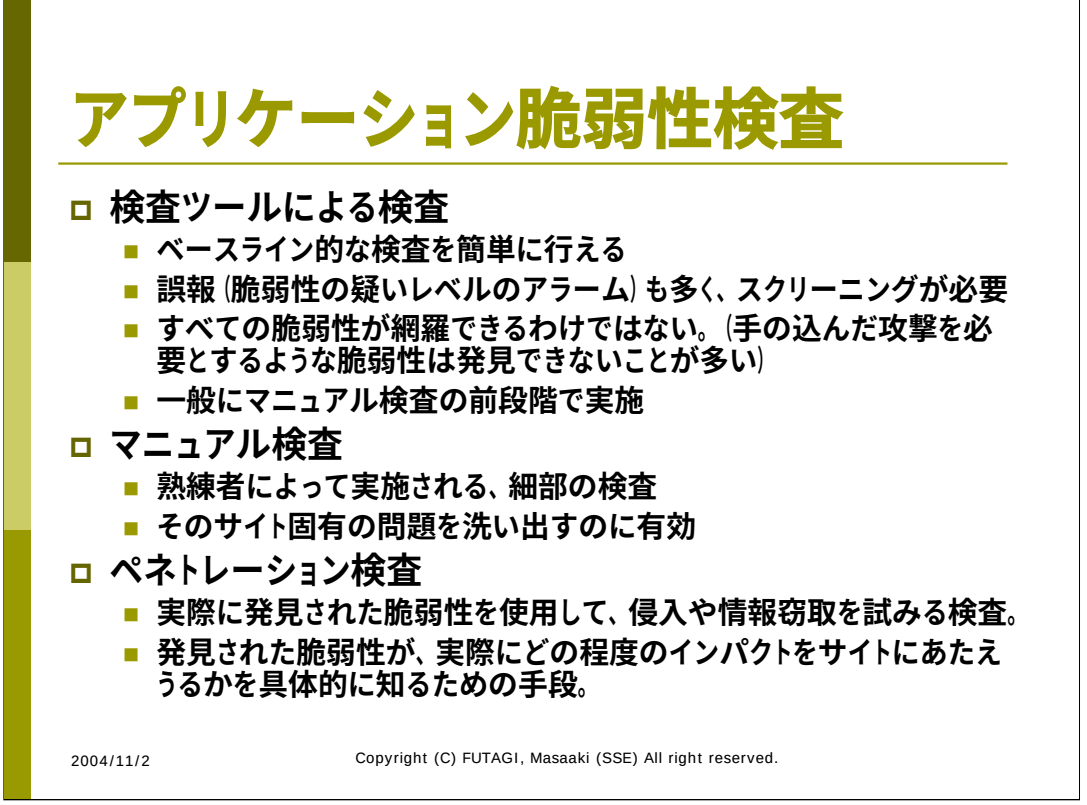
外部からアプリケーションの問題を調べるには、脆弱性検査を行うのがよい。

考えられるケースとしては

･すでに運用されており、セキュリティが十分に考慮されていない可能性があるアプリケーション
･新しく納入されたアプリケーションの受け入れ検査をかねて
･アプリケーション改修後の確認のため

では、実際に脆弱性が発見された場合、どうするか････。

なかなか教科書通りにおかないことも多い。改修できないアプリケーションをどう保護するかという視点も必要。
監視、や防御策としてのアプリケーションF／W（後述）など



脆弱性検査にはいくつかのレベルがある

・検査ツール

既知のプラットホームの脆弱性と基本的なアプリケーションの脆弱性について検査が可能。但し、あくまでベースラインであり、特にアプリケーション自体の脆弱性については、誤認（もしくは、たとえば「疑いあり」といったアラーム）も多く、最終的には人手による確認作業は必要になる。

・マニュアル検査

熟練者がそのノウハウを駆使して検査を行う。検査を簡素化し、網羅性を高めるために検査ツールによる検査を前もって行ってから、それを補完する形で行うケースが一般的。サイト固有の問題を洗い出すのに有効。

・ペネトレーション検査

マニュアル検査の一種とも言えるが、どちらかといえば「網羅性」よりも、見つけた脆弱性をどこまで使えるか、何ができるか、といった点にフォーカスして、実際に攻撃を受けた場合のインパクトを見極めるような検査方法をとる。実際のリスクを評価するための手法。

脆弱性を発見したら

- 「すぐに修正する」が基本である。
- しかし、現実には修正できないケースも多々ある・・・・
  - 古いアプリケーションで、開発委託先が対応できない
  - 非常に重要なアプリケーションのため、充分なテストを行ってからでないとリリースできない（したくない）
  - 修正していたらリリース期限に間に合わず、それによって大きな利益の逸失が生じる
  - ・・・・などなど
  - でも、無策はダメ。少なくともなんらかの代替手段を！

2004/11/2

脆弱性を見つけたら

とにかく、すぐ治せが教科書的回答なのだが、現実には難しいことも多いというのは、パッチあてよりも、こちらのほうが深刻化もしれない。

修正出来ない事情は様々でやむを得ないかもしれないが、少なくとも「無策」では、路上に金庫をおいておくようなもの。

検知が困難なアプリケーション攻撃

- アプリケーションへの入力内容の監視
  - 一般のIDSなどでは簡単ではない。(特定のパラメータのみを監視するなどアプリケーションの挙動のレベルでの追跡が必要。プロトコルレベルでしか追跡できないIDS/IPSでは誤報が多く発生する可能性が高い。)
- セッション単位での監視が必要
  - サーバから送られた内容が改ざんされて戻されたかどうか・・・などを調べる必要性。
- たんねんなログ解析などから発見できる可能性はある
  - リアルタイム性に欠ける・・・・

2004/11/2

実際に、アプリケーションに対する攻撃をリアルタイムに発見するのは、なかなか難しい。

IDSなどでは、アプリケーションの挙動までは認知できないので、このようなレベルの攻撃に対するシグネチャを作ることが難しい。作れても、誤認がかなり多くなると考えられる。

アプリケーションのセッション単位の監視が必須。前のコネクションで送られた内容について、次のコネクションで改ざんを調べるといったことは現状のＩＤＳでは不可能。

ログを見れば、ある程度判断はできるが、あとから調べることになるのでリアルタイム性はない。

外部的な保護手段

- アプリケーションファイアウォールとは
  - アプリケーションを攻撃から保護するためのしくみ
  - 様々なアプリケーション攻撃を、その手段から封じる
    - 入力のチェックと不正な文字、キーワードの排除
    - コンテンツへの直接アクセスの制限
    - 認証機構やセッション管理の強化
    - Cookie やセッション情報の保護（隠蔽、改ざんチェックなど）
    - SSL（アクセラレーション） のサポート
    - その他、一般のファイアウォールが有する機能・・・
  - 設定、運用が煩雑になりがちな点が問題
    - 個々のアプリケーションに固有のポリシーが必要
    - 検査ツールとの連携や、より抽象的な設定ができるように（よりインテリジェントに）なることが課題

2004/11/2

最近、Webアプリケーションの脆弱性を保護するようなファイアウォール製品が出始めている。一種のリバースプロキシとして動作し、クライアントからの送信を検査するようなタイプのものだ。

基本的には、ここに書かれたような機能だが、XML Webサービスの監視や、負荷分散装置の機能、一般のファイアウォールと同等の機能をベースに持つものもある。今後注目される分野の製品。

アプリケーション固有の設定項目がどうしても多くなるので、ポリシー設定が煩雑化しやすいのが難点。アプリケーションの挙動を学習したり、検査ツールと連携したりして、自動的に、ある程度の設定を行えるものもあるが、このあたりは今後の課題。

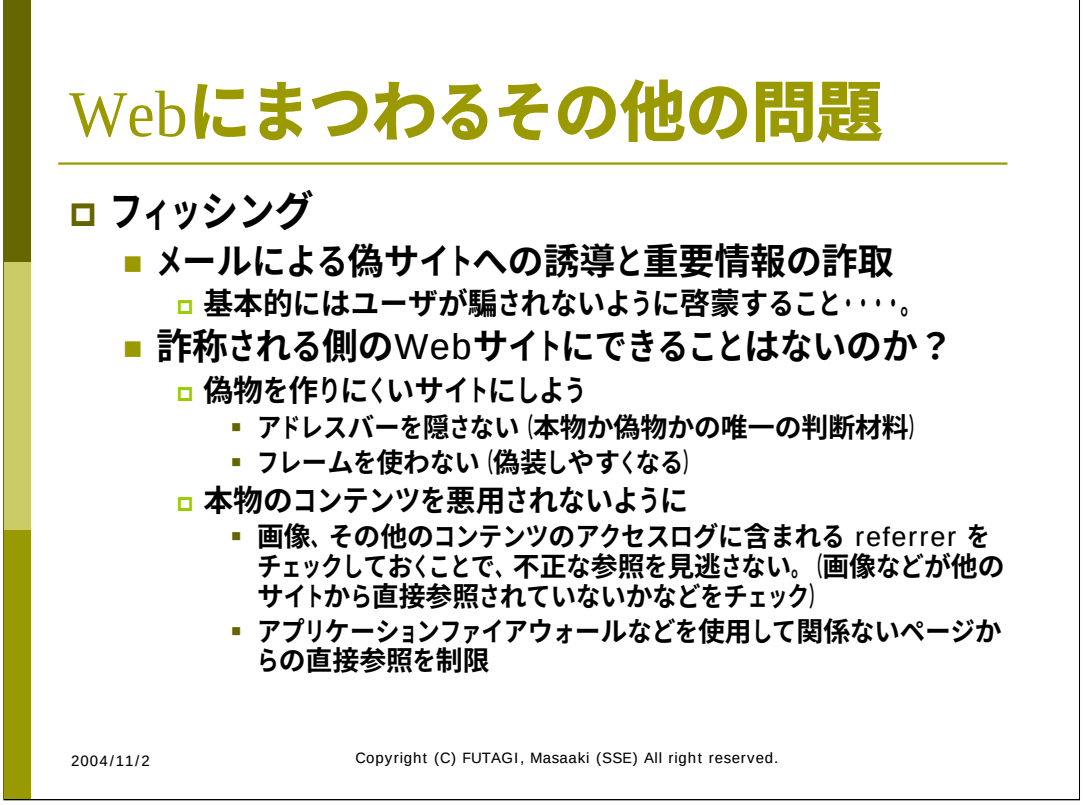
アプリ攻撃防止についてのまとめ

- 脆弱性の発見･排除
  - 開発レベルでの発見と排除（これが基本）
    - セキュアな設計とプログラミングの徹底
    - セキュリティを考慮したテスト
    - 設計、プログラミングのレビューの実施
  - 導入～運用時における発見と排除
    - 適切な検査の実施
    - ログ等の定期的なチェックによる攻撃、兆候の発見
    - 発見された脆弱性の修正（修正依頼、指示）
    - 外部的保護策の実施

2004/11/2

まとめ

一義的には開発マター（品質管理）だが、運用サイドでの対策もあわせて考えておいたほうがよい。



最近の話題から「フィッシング」

これは、ユーザがだまされる「詐欺行為」なので、詐称されたサイト側も被害者だと思われているが、はたしてそうなのか。たとえば、サイト側がちょっと気をつけることで、フィッシングに使われる偽サイトを作りにくくしてしまうこともできる。

たとえば、ブラウザのアドレスバーを消してしまうようなサイトもあるが、これは詐称を誘発して非常に危険。なぜなら、詐称サイトを判別する「唯一」に近い手段をユーザから奪うことになる。

フレームは、その中に表示されているものが実際はどこにあるのかわかりにくくしてしまう。フレームのプロパティを見ないとわからないので、たとえば、悪意サイトのスクリプトから、本物サイトを新しいブラウザウインドウで開き、さらに一部のフレームの内容のみを偽物にしてしまう、といったことが理論的には可能。たとえば、あたかも○×銀行のページだが、ログインフレームだけは、偽物・・・・といったようなことができてしまう可能性も・・・

このような本物コンテンツの悪用は、サイト側も目を光らせておくべき。ログにreferrer を取得しておくと、間接的なページ参照や、一部のコンテンツの直接参照は見つけやすい。不審な参照元を見つけた場合は、なんらかの対応、対抗手段をとることも、ユーザやビジネスの保護の観点から必要だろう。

Webにまつわるその他の問題

- ブラウザの脆弱性を悪用した攻撃
  - Download.jectの場合
    - IE の cross domain 脆弱性 (CAN-2004-0549)を悪用
    - 不正なスクリプトが仕掛けられたサイトをアクセスすると自動感染
  - Webサイトを見るだけで感染する悪性プログラム
- 攻撃側にとっては踏み台サイトの確保が必要
  - 踏み台サイトにならない。(全般的なセキュリティの確保)
  - 詐称・誘導 (フィッシング) のネタにならない
    - 模倣、偽造しにくいサイト作りを

2004/11/2

ブラウザの脆弱性を悪用した攻撃は、多くの場合、悪意のサイト (多くが踏み台) が必要。当然、自分のサイトの管理をサボったために踏まれてしまい、不正なものを組み込まれてしまったら洒落にならない。

また、フィッシングのように自分のサイトに似せて作ったサイトにユーザを誘導される可能性もあるので、フィッシング同様にこのような行為をどう防ぐかという点もサイトにとっては重要。

まとめ

- Webサイトへの脅威
  - 攻撃者の興味の対象はコンテンツへ
  - 特定の目的を持った攻撃の増加
  - 発見しにくい「アプリケーション攻撃」
- Webアプリのセキュリティ
  - 本質的には「セキュアな開発」の問題
    - 開発者のセキュリティ意識の向上を
  - 現実的には多段階の対策が必要
    - いかにして脆弱性を見つけ、潰すか・・・または攻撃を阻止するか
- クライアントへの攻撃、Webサイトにできることはないかを考えよう。

参考文献・資料

・IPA セキュアプログラミング講座
http://www.ipa.go.jp/security/awareness/vendor/programming/index.html

・IPA アクセス制御機構の機能不全を検出・検証するシステム
(株) ソフテック、 独立行政法人産業技術総合研究所 高木浩光氏
http://www.ipa.go.jp/security/fy14/development/web-auth/test-tool.html

・OWASP Top 10 日本語訳 (高橋 聡氏 訳)
https://sourceforge.net/project/showfiles.php?group_id=64424&package_id=70827

・OWASP Guide to Building Secure Web Applications
http://www.owasp.org/documentation/guide/guide_downloads.html

・安全なWebアプリ開発40箇条の鉄則
独立行政法人産業技術総合研究所 高木浩光氏
http://staff.aist.go.jp/takagi.hiromitsu/#2003.6.19
http://java-house.jp/~takagi/paper/idg-jwd2003-takagi-dist.pdf

ご清聴ありがとうございました

- 二木 真明
- 住商エレクトロニクス株式会社

2004/11/2