☞　早めのチェック　☜
3工程によるセキュリティ品質確保

「第2回 設計工程における考慮が重要である対策」

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〕

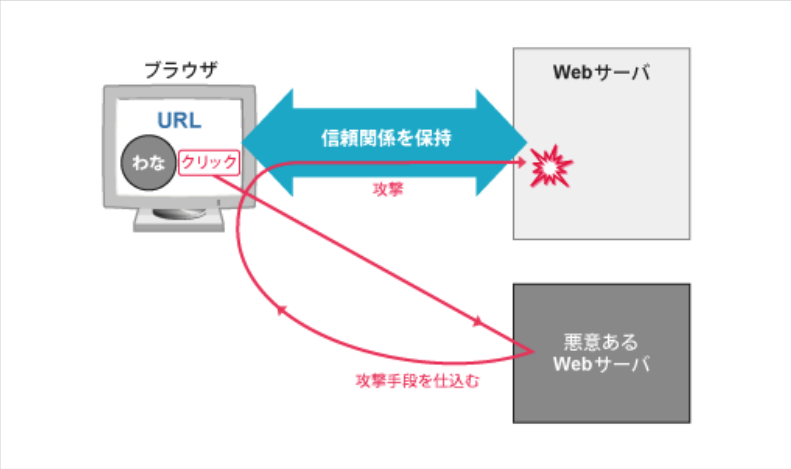
アジェンダ

2-1 セッション対策

2-1-1 リクエスト強要 (CSRF) 対策

リクエスト強要 (CSRF)

- リクエスト強要（CSRF）とは
  - Cross-site Request Forgery
  - 別に用意したコンテンツ上の罠のリンクを踏ませる等して、インターネットショッピングの最終決済や退会等Webアプリケーションの重要な処理を呼び出すようユーザを誘導し、被害を及ぼす攻撃



リクエスト強要 (CSRF) 攻撃対象

- 被害を受けるおそれがあるのは、次のようなWebアプリケーションのユーザ
  - Cookieを用いてセッションIDを搬送しているWebアプリケーション
  - Basic認証を用いているWebアプリケーション
  - Digest認証を用いているWebアプリケーション
  - そのほか、Webブラウザが自動送出する情報にもとづいてユーザやセッションを識別しているWebアプリケーション

リクエスト強要 (CSRF) 攻撃

- Cookie等の自動送出メカニズムを悪用
  - ブラウザが正規のWebコンテンツにアクセスする際は、所定のCookieやHTTP認証データがHTTPリクエストに付加され、Webサーバへ自動で送出されるという性質を悪用
- 意図しないリクエスト送出
  - HTTPリクエストの対象コンテンツ(「行き先」)が正規のものでありさえすれば、CookieやHTTP認証データは自動送出される
  - Webアプリケーションが用意したものでない、偽のフォームやハイパーリンク(「偽の出発地」)から発せられたものであっても、自動送出される
  - サーバ側では、偽のHTTPリクエストと本物のHTTPリクエストの見分けがつかない
- 本人の意図しない更新処理
  - 対策が講じられていない場合、正規のセッションに属す正規のHTTPリクエストであるとして受付けられ、ユーザ本人の意志に反した更新処理が行われるおそれがある

リクエスト強要 (CSRF) 対策

- 対策のねらい
  - ユーザ本人以外の人物がねつ造したコンテンツにもとづき発せられたHTTPリクエストをWebアプリケーションが受け付けない
- 仕組み
  - 重要な更新処理の場面において、秘密の照合情報をWebアプリケーションとブラウザの間でやりとりし、第三者が用意した偽のコンテンツから発せられたリクエストを検出できるようにする
- 具体策の例
  - フォームを表示するプログラムは、他者が推定・模倣困難なランダム値をhidden項目として埋め込む
  - フォームデータを受信するプログラムは、所定のランダム値がフォームデータ内に含まれているか検査する
  - 所定のランダム値がフォームデータに含まれていなかったり、値が一致しない場合、更新処理を行わない

2-1-2 セッション乗っ取り――セッションIDと侵害手口

セッションIDとセッションID侵害手口

- セッションを維持する仕組み
  - HTTP（Hypertext Transfer Protocol）には、セッション、すなわち複数のWebページからなる操作の流れを維持するための仕組みが十分には備わっていない
  - Webアプリケーションエンジンあるいはアプリケーションそのものがセッションを維持する仕組みを実装することが多い
  - しばしば用いられる手順は次のようなもの
    - あるタイミングでサーバ側が識別子をブラウザに向けて発行する
    - ブラウザからサーバに送られるリクエストにはその識別子が含まれるようになっている
    - サーバはその識別子にもとづいてセッションを維持する
  - ここでは、この識別子のことを「セッションID」と呼ぶ

セッションIDの搬送手段

- セッションIDの搬送手段
  - Cookie
  - URLリライティング
  - hidden項目

搬送手段1: Cookie

- CookieによるセッションIDの搬送
  - Set-Cookieレスポンスヘッダを用いてサーバ側がCookieを発行し、そのCookieをブラウザが自動でサーバへ送り出す方式
  - プログラミングの手間が最も少ない
  - セッションIDの搬送に最も多く用いられていると考えられる
  - Cookieを発行する際に与える属性には次のような注意を払う
    - domain={Cookie返送先ドメイン}
    - path={サイト内のパスのプレフィクス}
    - max-age={有効期間(秒数)}　expires={満了日付}　（どちらか一方）
    - secure　（指定するかしないか）

搬送手段2: URLリライティング

- URLリライティングによるセッションIDの搬送
  - サーバからブラウザにHTMLドキュメントを送り出す際、そこに書かれているURLのそれぞれにセッションIDを含めるよう書き換えてから送出する方式
  - URLリライティングの方式では、URLの一部にセッションIDが含まれているため、キャッシュ、ログ、Referer:ヘッダ等を通じて第三者にその値が漏えいするおそれがある
  - URLリライティングは、Cookieが使用できない場面でやむを得ず使用する以外、使用を推奨しない

搬送手段3: hidden項目

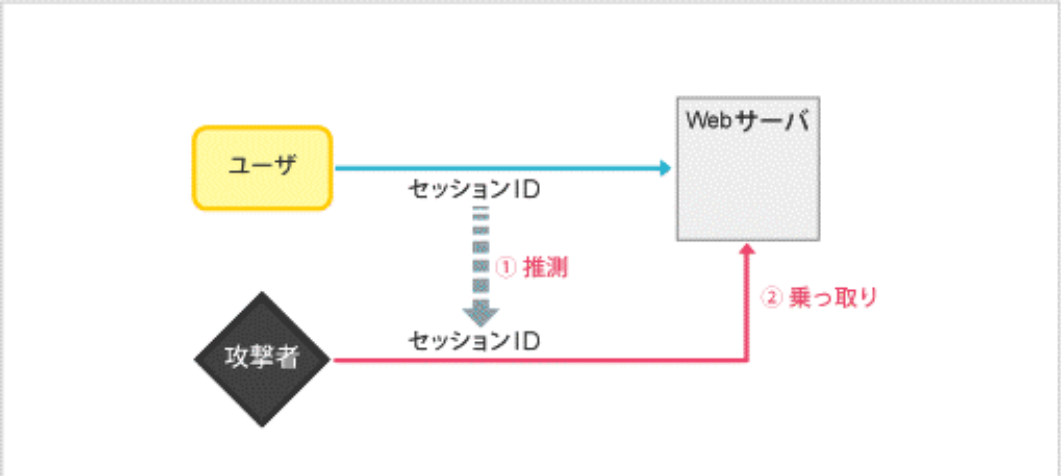
- hidden項目によるセッションIDの搬送
  - すべてのページ遷移をフォームデータの送信の形で記述し、フォーム中のhidden項目にセッションIDを含める方式
  - Cookie、URLリライティングに比べ、第三者に値が流出する事故が起こる要因は少ない
  - ただし、自然なハイパーリンクの実装のためにJavaScriptのコードを記述する必要が生じる

セッションID侵害手口の分類

- セッションID侵害手口の分類
  - セッション乗っ取りは、正規ユーザが用いているセッションIDを攻撃者が支配下におくことによって、攻撃者が正規ユーザになりすましてWebアプリケーションを操作するもの
  - 攻撃者が正規ユーザのセッションIDを支配下に置く手口には次の3種類がある
    - セッションIDの推測
    - セッションIDの奪取
    - セッションIDのお膳立て

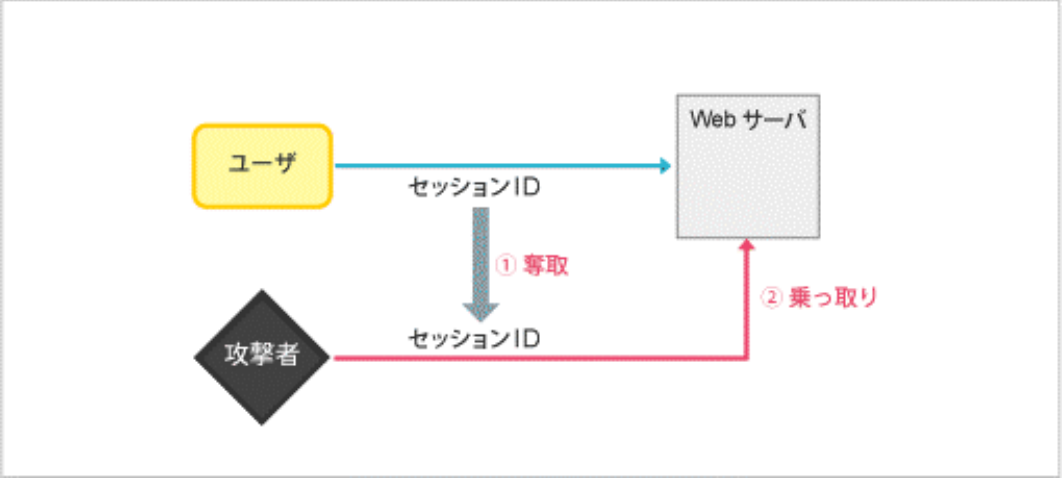
侵害手口1: 推測

- セッションIDの推測
  - 生成規則が複雑でないセッションIDの現在使われている値を試行錯誤で見つけ出す



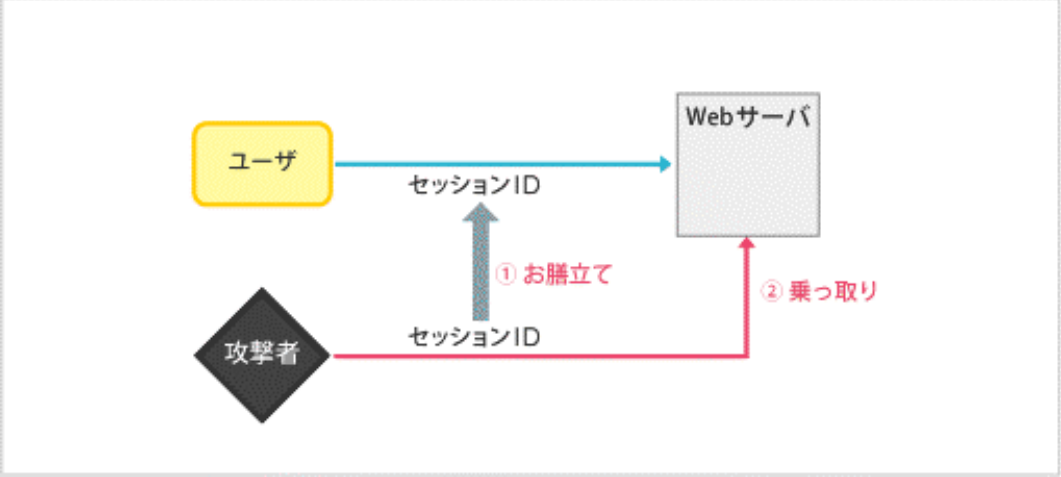
侵害手口2: 奪取

- セッションIDの奪取
  - ネットワーク盗聴やスクリプト注入を用いて、実際に使われているセッションIDの値を入手する



侵害手口3: お膳立て

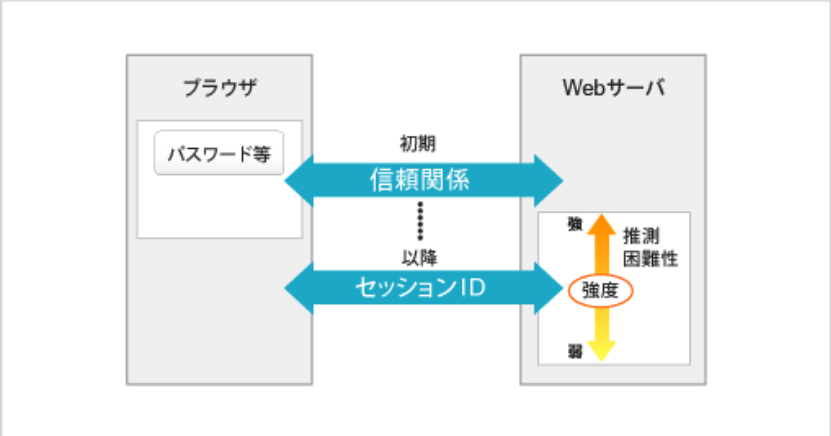
- セッションIDのお膳立て
  - 罠のハイパーリンクやスクリプト注入を用いて、被害者のブラウザにセッションIDを植え付ける



2-1-3 セッションID強度を高める

セッションID強度を高める

- セッションIDという値の必要条件
  - セッションIDは、Webページの遷移を維持し、必要な手続きを経て情報を閲覧あるいは、登録・変更・削除を行うための重要な値
  - 容易に推測されない(破られない)強度の高い値にすることが重要



セッションID強度を高める

- 人に知られていそうなIDは使用しない
  - 例えば、会員番号やユーザID等、他者に知られているおそれのある値は推測が容易であるので、セッションIDに使用すべきでない
- ランダムな値を使用する
  - 00001、00002…等の規則性や連続性のある値は、推測が容易
- 疑似乱数の慎重な選択
  - 自前でランダム値発生アルゴリズムを作らない
    - 多くの場合、ランダムさが足りない
    - 何かの値に対してハッシュ関数を使用しても、バリエーションが少なければ有効な値を推定されるおそれがある
  - 暗号論的擬似乱数生成器と呼ばれる種類の疑似乱数を使用する
    - ゲームやシミュレーション用の疑似乱数の使用を避ける

セッションID強度を高める

- 桁数と文字種を多くする
  - 桁数や文字種のバリエーションが少ないと、総当たり攻撃で破られる（正しい値を見つけ出される）おそれがある
  - 使用する文字種、桁数をなるべく多くする
  - 例
    - 英小文字・英大文字・数字の任意の10桁でセッションIDを構成する場合
    - 組み合わせの数は、(26+26+10)^10 ≒ 8.39×10^{17} 通り
    - 同時に1万件のセッションIDが有効な状態で存在し、Webサイトのリクエスト処理能力が、100万件/秒　であるとすると、総当たりで有効なセッションIDを1つ見つけ出すのに要する時間は平均でおよそ2年半
- 毎回異なる値を発行する
  - 一人のユーザのセッションIDがいつも同じ値であると、一度セッションIDの値を知った人物は、毎回セッションの乗っ取りができる

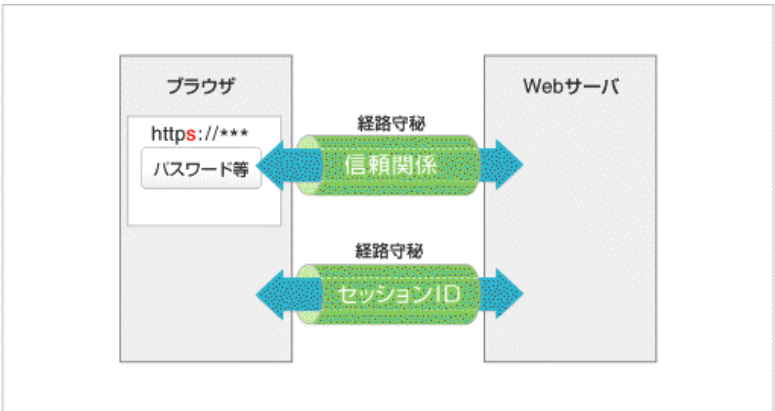
処理系が用意しているセッションIDの利用

- Java Servlet
  - セッションIDは JSESSIONID という名前のCookieで搬送される
- ASP
  - セッションIDは ASPSESSIONIDxxxxxx（ランダムな英字列が末尾につく）という名前のCookieで搬送される
- PHP
  - セッションIDは PHPSESSID という名前のCookieで搬送される

2-1-4 https:の適切な運用

https:の適切な適用

- https:による通信経路の防護
  - 他人に通信を傍受されたくない場面
    - パスワード入力、個人情報表示、クレジットカード番号の取り扱い等
  - 経路の守秘性を確保するためにセキュアプロトコルを使う
    - SSL (Secure Socket Layer) あるいは TLS (Transport Layer Security)
  - http: ではなく https: のスキームのURLが使われる



暗号通信で守るべきもの

- セキュアプロトコルで守るべきもの
  - ユーザが目にする機密情報
    - パスワード、個人情報、クレジットカード番号、ショッピング明細等
  - セッションID
    - ユーザの目に触れないところでWebアプリケーションはセッションIDをブラウザとやり取りしている
    - セッションIDが第三者に知られれば、セッション乗っ取りが生じるおそれがある
    - 暗号通信でセッションIDの秘密を守ることも、忘れてはならない対策のひとつ

セッションIDが漏れる

- セッションIDが漏れる
  - https: を使っていてもセッションIDが第三者に漏えいすることがある
- ケース1
  - ひとつのホストあるいはディレクトリの中で http: のページと https: のページが混在
    - https: の通信でのみ使うべき Cookie の値が http: の通信に流出するおそれ

【対策】https: のみの用途の Cookie には、発行時に secure 属性を付ける

- ケース2
  - ログイン前のhttp:のページですでにセッションIDが発行されている
    - このセッションIDは第三者に傍受されるおそれがある
  - ユーザがログインした後もログイン前と同じセッションIDが使われる
    - 第三者に傍受されたセッションIDである可能性がある

【対策】ログインするまでのhttp:のページではセッションIDを発行しないようにするか、またはユーザがログインに成功した時点でそれまでのセッションIDを無効にして新たなセッションIDを発行し直す

https: にまつわる諸対策

- 上流からのhttps:適用設計
  - 暗号通信で保護すべきページを見極める
  - 保護すべきと定めたページはhttps:でのみ閲覧させる
  - これらのページへのアクセスにあたって最初に本人認証を行う
  - 認証の有効性をセッションのメカニズムを用いて複数のページにわたり維持する
  - http:のコンテンツとhttps:のコンテンツは別のホストから提供する
  - https:のコンテンツのhttp:による閲覧を許さない。http:のコンテンツのhttps:による閲覧を許さない
- SSL、TLSはなるべく高いバージョンのものを使用する
  - SSL 3.0 以上、TLS 1.0 以上
- サーバ証明書を自己発行しない
  - 本番運用のWebサーバに関しては、サーバ側ディジタル証明書を自己発行すべきでない。
  - クライアント側でその証明書の正当性を検証しようとしても、当該Webサイトが提供するルート証明書のみにもとづいて検証することになる。
  - ルート証明書の段階から偽物を作ることで攻撃者は証明書の辻褄を合わせることができ、詐欺サイト等が作られるおそれがある。
- Cookieにsecure属性をつける

2-1-5 セッションIDのお膳立てへの対策

セッションIDのお膳立て

- セッションIDのお膳立て
  - セッション乗っ取りの手口のひとつ
  - 有効なセッションIDを攻撃者が確保し、それを被害者に使わせる
  - セッションフィクセーション(session fixation)とも呼ばれる

セッションIDお膳立てのメカニズム

(1) 隙のあるWebアプリケーションエンジン

- セッションIDはWebサーバ側が発行してブラウザに預け、Webサーバへアクセスするたびにその値をブラウザが送り出すのが原則
- WebアプリケーションやWebアプリケーションエンジンの中には、自分が発行したものではないセッションIDの値をブラウザが送ってきても、それを有効なセッションIDとして許してしまうものがある
- 代表格はPHPの処理系である。例えば、下記のURLのように、明示的にPHPSESSIDパラメータを付けてリクエストを送ってみる

  http://foo/bar.php?PHPSESSID=3333

- すると、このリクエストに対する応答のヘッダには下記の内容が含まれる。

  Set-Cookie: PHPSESSID=3333; …

- つまり、ブラウザから与えた値がセッションIDとして採用されてしまう。
- 攻撃者は上記のセッションIDを仕込んだハイパーリンクを踏ませることによって、被害者を罠にかけることができる。このセッションIDが使われたままユーザがログイン手続きを行うと、攻撃者もそのユーザのプライベートなページを見て回る機会を得る。

セッションIDお膳立てのメカニズム

(2) 堅いエンジンにも攻撃は可能

- 多くの処理系では、ブラウザが勝手にセッションIDパラメータの値を送ってきてもそれを正規のIDとして受け付けないことが多い
- しかしそれでもセッションIDお膳立て攻撃は生じ得る
- 攻撃者自身がWebアプリケーションから正規にセッションIDの発行を受けておき、そのIDの値を罠のハイパーリンクやスクリプト注入等の手口で被害者のブラウザ上のCookieに送り込む
- Webアプリケーションにスクリプト注入脆弱性が残っている場合、例えば、攻撃者は、次のような文字列がWebページに埋め込まれるように仕組んで被害者のブラウザのCookieに値を設定する

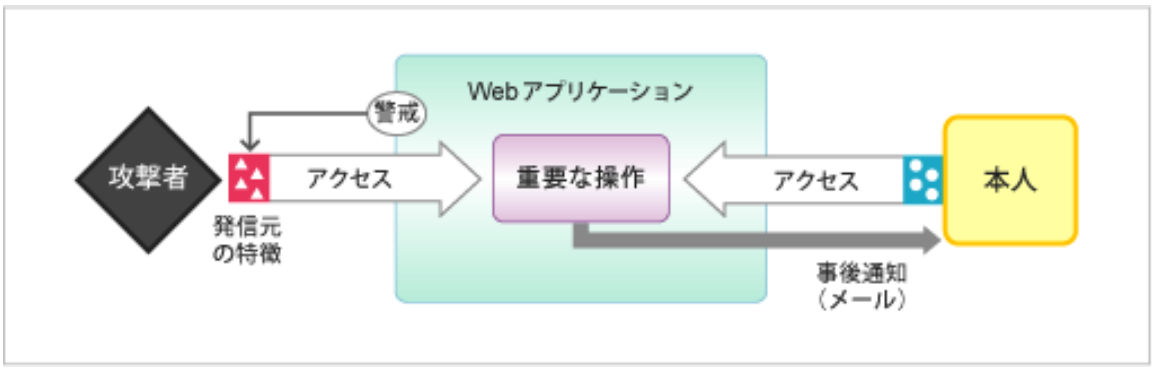
  <script>document.cookie="sessionid=foobar"</script>

セッションIDお膳立て対策

- セッションIDお膳立て対策
  - セッションIDお膳立てに対抗するには、セッションIDの付け替えを行う。この方法は、ユーザがログインに成功した時点でまったく新しいセッションIDを発行し、それまでのセッションIDを無効にするというものである。
  - ユーザ認証とセッションを取り扱うWebアプリケーションには、この「セッションIDの付け替え」の実装を強く推奨する。
  - また、スクリプト注入脆弱性対策はセッションIDのお膳立て対策としても必要不可欠である。
- 事例・アプリケーションエンジンの脆弱性
  - その後ベンダーによって対策が施されているものの、2004年 9月にはMacromedia JRunにセッションIDお膳立ての脆弱性が報告されている。
  - それまでのJRunは、ブラウザから送られてくるセッションIDパラメータJSESSION の値を無条件で受け入れてしまっていた。(CVE-2004-0646)
  - この脆弱性に対策を施さないままのWebサイトでは、セッションID のお膳立て手口によるセッション乗っ取りが生じ得る。

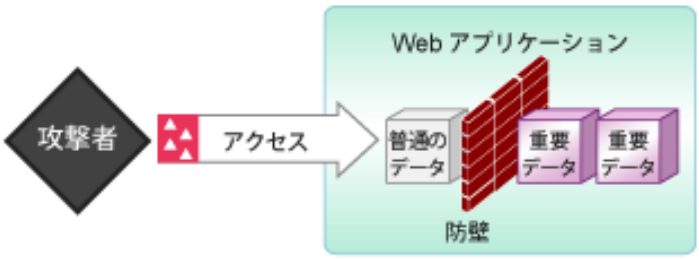
2-1-6 乗っ取り警戒と被害の不拡大

- 乗っ取り兆候の警戒(検知策)
  - クライアントから送られてくるリクエストの内容等について、正規ユーザと攻撃者の差異を見いだし、乗っ取りの兆候を検知できる場合がある。
- 乗っ取り兆候の警戒

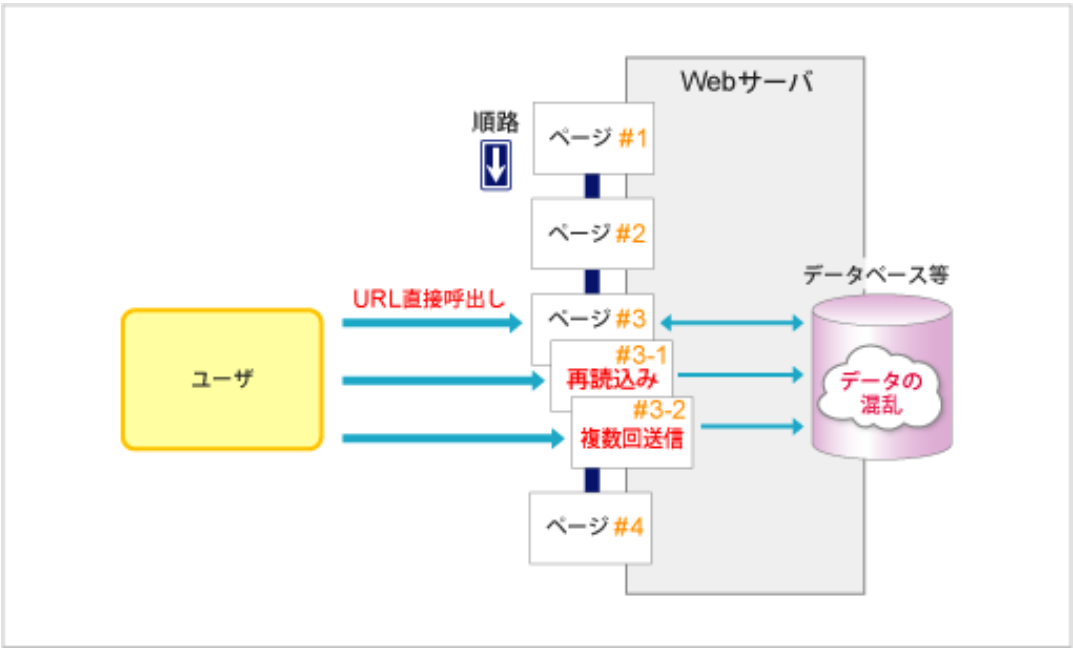


乗っ取り被害を拡大しない工夫(事後策)

- 乗っ取り被害を拡大しない工夫(事後策)
  - たとえ、セッション乗っ取りが生じたとしても、すべてのWebページのアクセスをユーザに許すのではないなんらかの歯止め策があれば、被害をある程度食い止めることができる。
- 乗っ取り被害を拡大しない工夫



2-1-7 想定外ナビゲーション対策



2-2 アクセス制御対策

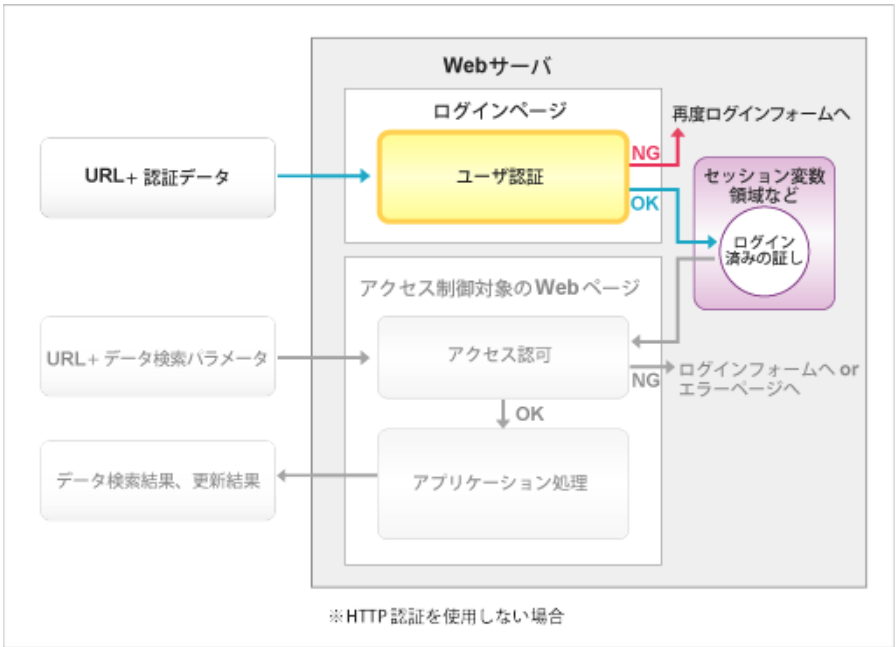
2-2-1 ユーザ認証対策
2-2-2 アクセス認可対策

2-2-1 ユーザ認証対策

アクセス制御

- アクセス制御
  - 個人情報、取引情報、有償コンテンツ等を取り扱うような場合、Webアプリケーションにおいても、リソースへのアクセスを許可されたユーザのみが行えるようにする仕組み、すなわちアクセス制御のメカニズムが必要
- アクセス制御は2段階で構成
  1. ユーザ認証
     - アクセスしてきた人物が本人であることを確かめる手続き
  2. アクセス認可
     - 認証されたユーザに、コンピュータのリソースへのアクセスを許可あるいは禁止する仕組み
- 必ずサーバ側で行う
  - アクセス制御の処理は、必ずサーバ側で行うことが重要
  - クライアント側で行うと、ユーザは制限を迂回できる

Webアプリケーションにおける ユーザ認証の概要



アカウントのロックアウト

- ログイン失敗の連続を放置することの問題
  - 無制限にログイン失敗を許容してしまうと、パスワードやユーザIDを総当りで検証する攻撃「ブルートフォースアタック」により、認証を突破されてしまうおそれがある
- 対策：アカウントのロックアウト
  - 一定時間内のログイン失敗回数が基準値を超えた場合には、そのアカウントを使用禁止にするべきである
    - 例えば、あるアカウントに対するログインが10回連続して失敗した場合に、そのアカウントのログインを数時間禁止する
    - ロックアウトされるログイン失敗回数やロックアウト期間は、利用上の利便性との兼ね合いにより設定する
  - 警戒すべきパターン
    - 同じユーザIDに対する異なるパスワードの試行
    - 同じパスワードに対する異なるユーザIDの試行
  - ロックアウトしたことそのものを知らせない
    - 「ロックアウトした」旨のエラーメッセージは表示すべきではない
    - 試行したユーザIDが実在することを攻撃者へ明かさないように

パスワードフィルタ

- 強度の低いパスワードの問題
  - 類推、辞書攻撃、総当り攻撃で容易にパスワードが盗まれてしまうおそれ
    - パスワードがユーザIDと同じ、一般名詞を使用している、文字数が少ない、文字種が英字小文字のみ　等
- 対策：パスワードフィルタ
  - 強度の低いパスワードをユーザが設定しようとした場合に、そのパスワードの採用を拒否するしくみ
  - あらかじめ決めた規則に従ってパスワードの採用の許可・不許可を判定する
- パスワードフィルタの規則の例
  - ユーザIDが含まれるものは許可しない
  - 生年月日が含まれるものは許可しない
  - 前回と同じパスワードは許可しない
  - 8文字以上で英字と数字と記号がどれも含まれるものを許可する　等

ログインエラーメッセージ

- 親切すぎるログインエラーメッセージの問題
  - ログイン失敗のエラーメッセージに要因の説明が含まれていると、攻撃の手がかりを与えることになる
  - 数組のユーザIDとパスワードを用いてログインを試行し、以下のメッセージが表示された場合
    - 1組目の試行へのエラーメッセージ:「ユーザIDが正しくありません」
    - 2組目の試行へのエラーメッセージ:「パスワードが正しくありません」
    - 3組目の試行へのエラーメッセージ:「ユーザIDが正しくありません」
  - これらのメッセージから判明すること
    - 1組目と3組目の試行のユーザIDはアカウントが存在せず、2組目の試行で入力したユーザIDのアカウントは存在する
  - ログイン場面に限ってはエラーメッセージの説明については、あいまいにし、具体的に何を間違えたのかをユーザに伝達すべきではない
- 対策:何を間違えたかを具体的に教えないエラーメッセージ
  - 例 「ユーザIDまたはパスワードが違います」

そのほかの考慮事項

- ユーザIDのみで認証を行わない
  - まれに、パスワードを用いないサイトがある → 脆弱
  - 常にパスワードの入力を求める
- パスワードの有効期限と変更
- ランダム化桁とチェック桁
- パスワードリマインダの慎重な設置
- 管理者権限の制限
  - 他者のパスワードを見ることができないようにする

2-2-2 アクセス認可対策

アクセス認可対策

- アクセス認可
  - 認証済みのユーザに対しデータやプログラム機能等のコンピュータのリソースについての利用を許す・許さないを具体的に制御する手順
- アクセス認可の実現の際に留意すべき点

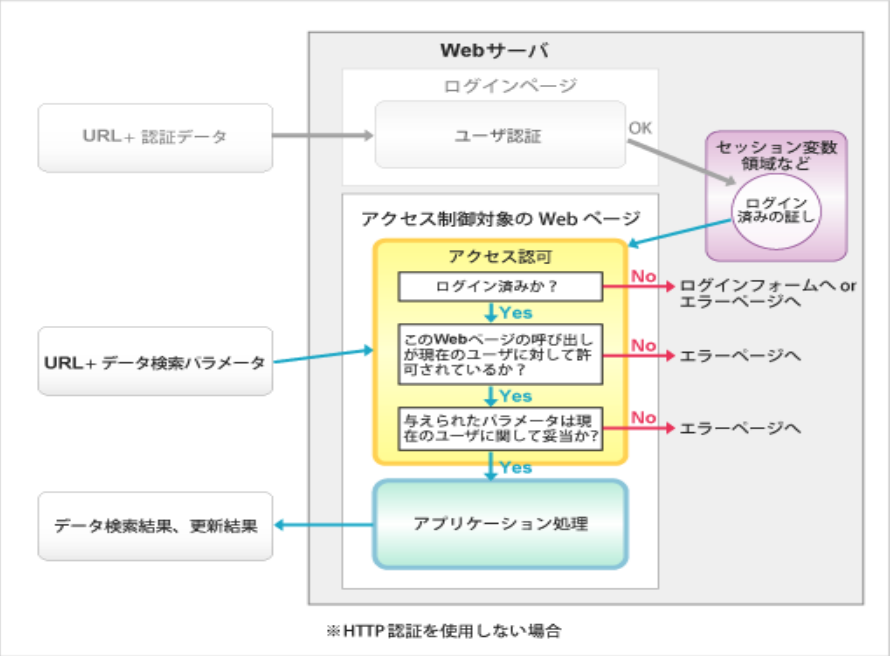
(1) 認証にもとづいて行う

- アクセス認可は、あらかじめユーザ認証（本人認証）を行っておき、その結果にもとづいて行う必要がある
- クライアントの接続IPアドレスや、パスワードの提示を受けずに受け入れたユーザ識別番号等を用いてアクセス認可を行う方法は、他人によるなりすましを見過ごすおそれがある
  - ユーザを特定することが重要でないシステムを除き、これらの方法は避けた方がよい

(2) ユーザの全アクセスをインターセプトする

- ユーザから対象のリソースへのアクセスが起こる際、アクセス認可のためのロジックを迂回できては具合が悪い
  - 例えば、選択できる項目の表示をメニュー画面上で隠したのみでは、URLを用いて直接リソースが呼び出せるといった問題があり得る
- ユーザがリソースにアクセスする経路に立ちふさがる形で仕組みを設ける
  - ユーザがアクセスを試みたときには必ずアクセス認可の判定ロジックがはたらくようにする

アクセス認可の三段階のチェック



ログイン有無による制限

- ユーザがログイン済みでなければコンテンツを開示しない
  - 会員制のサイトや有償コンテンツを提供するサイト等
    - 特定のページはログイン済みのユーザでなければ閲覧を制限するような仕組みが必要
  - 制限の対象となるWebページを表示する各プログラムにおいて
    - 現在アクセスしているユーザがログイン済みであるか否かをプログラム冒頭で判定する

ページ単位のアクセス許可・禁止

- 各ユーザごとに閲覧の許可・禁止を制御したいWebページ
  - ログイン済みのユーザに無条件ですべてのWebページの閲覧を許してはまずいケース
  - 例えば、オプションの申し込みに応じて利用できる内容が変化するサイト、一般会員と特別会員のように会員に複数の種別があるサイト等
- そのようなWebページを表示する各プログラムにおいて
  - 現在ログインしているユーザの識別子と表示しようとしているページの組み合わせが「許可されている」ものであるか否かを判定するロジックをもつ
    - 組み合わせが「許可されない」ものであればコンテンツを開示しない
- おおよその仕組み
  1. どのユーザにどのページの閲覧を許すかを示すデータ
  2. 現在ログインしているユーザと閲覧しようとしているページの組み合わせが「許可されている」か否かを判定するモジュール
  3. 各Webプログラムでは、ユーザがログイン済みであることを判定した後、この共通モジュールを呼ぶ

パラメータ単位のアクセス許可・禁止

- パラメータ単位の制御
  - あるいは「オブジェクト」単位の制御
  - そのWebページそのものの閲覧はどのログイン済みユーザにも許すが、ユーザには権限のあるリソースのみアクセスできるように制限を加えるケース
- 不都合な例
  - 会員のショッピング履歴項目のひとつひとつを一意に識別できるキーの値が存在し、それをWebプログラムのパラメータに与えるようになっている場合
    - このパラメータにうまく値を与えると他人のショッピング履歴が不正に参照できるとしたら具合が悪い

うまくいかない実装の例

- アクセス認可のうまくいかない実装方式の例
  (1) 入力パラメータでユーザを識別する
    - 攻撃者は、そのパラメータを偽造する。
  (2) メニュー画面からリンクを隠すのみ
    - 攻撃者は、URLを直接指定してページを呼び出す。
  (3) Referer: ヘッダを見て制御する
    - 攻撃者は、ツールを使って Referer: ヘッダを偽造する。
  (4) GETメソッドに反応せずPOSTのみに応答を返す
    - 攻撃者は、ツールを使ってPOSTリクエストを投入する。

2-3 総論・その2

2-3-1 より良いWebアプリケーション設計のヒント

2-3-1 より良いWebアプリケーション設計のヒント

より良い設計のヒント

- より良いWebアプリケーション設計のヒント
  - 以降のスライドで述べるのは、脆弱性が生まれにくいWebアプリケーションを構築するために設計段階、あるいはそれ以前の段階で考慮しておくとよい事項の例である
  - 次の7つの場面について述べる
  - 開発基盤選定
  - サイト配置設計
  - アクセス制御設計
  - セッション設計
  - ログ記録設計
  - 業務仕様設計
  - モジュール分割設計

開発基盤選定における考慮事項の例

(1) プログラミング言語の選択

- 例えば、PHPを避ける
- 大規模システムにはJavaや.Netを使う

(2) データベースAPIの振舞い把握と製品選択

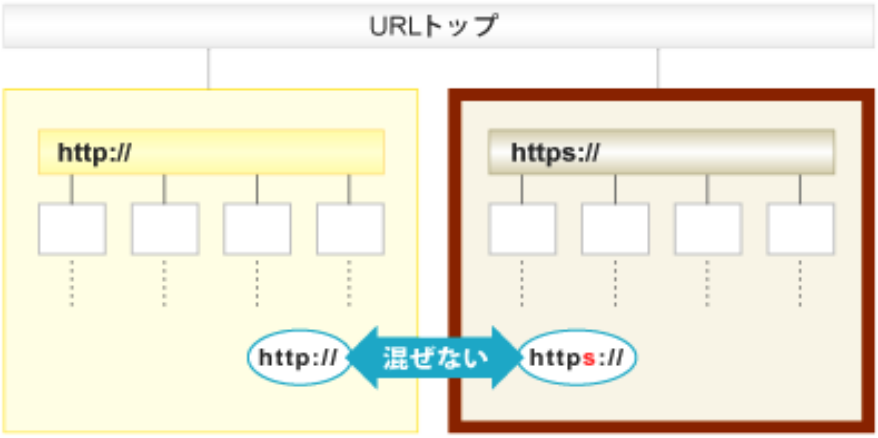
- DBはSQL注入攻撃の危険に曝されている
  - SQLを用いている限り常に
- DBアクセスAPIがどのような特徴を持っているか把握しておく
  - プリペアドステートメントが使えるか？
  - どのような特殊記号エスケープ文法を持っているか？
  - マルチバイト文字は何をサポートしているか？
  - マルチバイト文字の非標準のビットパターンを受け入れてしまうか？
  - マルチバイト文字の一部に特殊文字に相当するビットパターンが含まれているとき、マルチバイトと特殊文字のどちらに反応するか？　等

サイト設置設計における考慮事項の例

- 方針＝サイトの用途別分離
  - 用途の異なるコンテンツがひとつのサイトに混在しないよう、複数のサイトへの分離をはかる
- 例えば
  - http:専用サイト と https:専用サイト を分ける
    - 同じサイト上にhttp:ページとhttps:ページが混在しないようにできるのであれば、https:専用サイトのCookieが平文でネットワークに流出する問題を防ぎやすい
  - 商品カタログサイト と ショッピングカートサイト を分ける
  - 会員向けサイト と 管理者向けサイト を分ける　等
    - あらかじめサイトを分離しておくことで、アクセス制御（本人認証とアクセス認可）のロジックが複雑になるのを避けることができる。これは、アクセス制御が迂回される脆弱性を起こりにくくする効果がある。

サイト配置設計・2

- 目的ごとにサイトを分ける



セッション設計における考慮事項の例

(1) 前後関係の追跡とログイン状態の追跡

- ログイン前からセッションIDを用いる必要があるか否かを明らかにする
  - 複数ページからなる会員登録フォームにおいてページの連鎖を追跡する等
- ログイン前からセッションIDを用いるのであれば、このセッションIDの取り扱いに加えて、ログイン状態を管理するしくみをもつ必要がある
  - ログイン前後におけるセッションID値の変更　または
  - ログイン状態を追跡するためのもうひとつのセッションID――「ログインセッションID」

セッション設計・2

(2) ログイン維持の方式

1) HTTP認証を使用する場合
- HTTPリクエストにユーザ認証データが含まれているため、特別なロジックは不要
- ユーザ認証データがやりとりされる通信にはすべてhttps:を使用

2) ログインフォームとセッションIDを使用する場合
- ユーザがログインに成功したら次を行う。
  - それまで使っていたセッションIDを無効にし、新しいセッションIDを発行
  - Webアプリケーション動作環境が提供してくれるセッション変数メカニズムを用い、セッション変数に「ログイン済み」であることを記録
  - セッションIDがやりとりされる通信にはすべてhttps:を使用

3) ログインフォームと、セッションID、およびログイン追跡パラメータの3つを使用する場合
- ユーザがログインに成功したら次を行う。
  - ユーザがログイン済みであることを示す、第三者が予測困難であって内容の解読が困難な値――すなわち暗号や乱数――からなるログイン追跡パラメータを発行
  - ログイン追跡パラメータはCookieやhiddenパラメータの値としてブラウザに預け、サーバに返送されるようにする
  - ログイン追跡パラメータをやり取りする通信にはすべてhttps:を使用

セッション設計・3

(3) リクエスト強要(CSRF)対策

– ログインセッションごとに単一の照合値を使う方式
  a. ユーザがログインに成功した時点で次を行う
    – リクエスト強要（CSRF）対策の照合値を乱数等を用いて生成し、セッション変数等に記憶させておく
  b. 重要な処理を行う直前の場面で次を行う
    – この照合値をフォームに埋め込んでブラウザに送る
  c. 重要なリクエストを処理するプログラムで次を行う
    – 送られてきたリクエストに同一の照合値が含まれているか否かを判定
    – パラメータが含まれていないか値が合致しない場合は重要な処理を行わない
– ページを呼び出すごとに異なる値をとるトークンを使う方式
  b. 重要な処理を行う直前の場面で次を行う
    – 照合値を乱数等を用いて生成し、セッション変数等に記憶させる
    – 毎回異なる値を用いる
    – この照合値をフォームに埋め込んでブラウザに送る
  c. 重要なリクエストを処理するプログラムで次を行う
    – 送られてきたリクエストに同一の照合値が含まれているか否かを判定
    – パラメータが含まれていないか値が合致しない場合は重要な処理を行わない

セッション設計・4

(4) https:適用範囲

– https:は、ログインページや個人情報入力・表示ページのみならず、ログインセッション中、常に使用する

1) HTTP認証を使用する場合
   - ユーザ認証データがブラウザからサーバへ送られるすべての通信でhttps:を使用する

2) ログインフォームとセッションIDを使用する場合
   - ログインの時点で新しい値が与えられたセッションIDがやりとりされるすべての通信でhttps:を使用する

3) ログインフォームと、ページの前後関係を維持するセッションIDおよびログインセッションIDの2つを使用する場合
   - ログインの時点で発行されたログインセッションIDがやりとりされるすべての通信でhttps:を使用する

(5) ログイン前セッションフィクセーション対策

– ログイン前のセッションの「セッションIDのお膳立て」攻撃への対策

ログ記録設計における考慮事項の例

- ログ記録方式の設計
  – どのようなイベントを記録するか
  – どの情報を記録するか
  – 記録する媒体を何にするか
  – ログの保護策としてどのようなものを講じるか
  – 複数コンピュータの時計の同期をはかる

業務仕様設計における考慮事項の例

(1) トランザクションに関して

- 入出力項目設計
  - サーバとブラウザの間でやり取りされるデータ項目を、HTTPプロトコル・レベルにおける姿で列挙し、仕様を記述する
  - これにもとづいて入力検査を行うことになる

(2) ユーザインタフェースに関して

1) ページレイアウト

- 本物であることの確認手段を封じないウィンドウレイアウトデザインをする
  - フレーム多用の回避
  - 目立つ場所への「ログアウト」ボタンの配置
  - ブラウザのアドレスバーおよびステータスバーを非表示にしない
  - マウス右ボタンコンテキストメニュー等、ページのプロパティを参照する手段を使用不能にしない

2) エラー時のページ遷移

- エラーの種類に応じた画面遷移を設計する
  - 未認証 → 「ログインしてください」、ログインページへ
  - 本人認証失敗→「ログインできません」、ログインページへ
  - アクセス認可違反→「このページ／リソースにはアクセスできません」、メニュー等へ
  - リクエスト強要の疑い→「順路から外れています」、メニュー等へ
  - 受け入れられない入力データ→エラーメッセージ、元の入力フォームへ
  - プログラム内部の異常→異常のお詫び、メニュー等へ

業務仕様設計・2

(3) コードに関して

1) コードの積極的使用

- チェックボックス、ラジオボタン、選択肢項目等、ユーザが直接値を入力しないフォーム項目も、悪意あるユーザによって値が改ざんされているおそれがある
  - これらの選択項目については、次のように取り扱う。
  - とり得る値について数字や英字からなるコードを割り当てる
  - HTMLタグにはコードの値を埋め込む
  - 入力の時点でコードの妥当性を検査する
  - 名称表示の必要があればテーブルを引いて名称を得、それを表示する
- これにより、外部からプログラムに不正な値が送り込まれる機会を減らすことができる。

ご質問をどうぞ

Q & A