Session Adoption in Session Fixation

ＮＴＴ コミュニケーションズ株式会社
ＩＴ マネジメントサービス事業部
セキュリティオペレーションセンタ

2010 年 03 月 09 日

Ver. 1.0

Copyright 2010 NTT Communications Corporation

1. 調査概要

現在の Web アプリケーションのほとんどは、ID(SessionID と呼ばれる識別子)を用いた HTTP セッション管理を行っている。
通常、SessionID は Web アプリケーション側で決定するが、クライアント(Web ブラウザ)側で決定した任意の値を SessionID として使える場合がある。
PHP はその代表例といえるが、このような脆弱性は「Session Adoption」問題と呼ばれている。

「Session Adoption」問題は「Session Fixation」問題の一部と考えることができる。
また、「Session Adoption」問題がどのような現象であるかの概要程度の記述はインターネット上にも散見されるが、実際のセキュリティ上の脅威としてはどの程度であるかといった検証レポートが見つからなかった。
本文書では「Session Adoption」問題が、どのように「Session Fixation」問題さらには「Session Hijack」問題と関係していくのか、また Web アプリケーションでどの程度の脅威となるのかを検討した。
さらに防御方法、特に Web アプリケーションのプログラミング方法についても検討した。

2. SessionIDによるHTTPセッション管理とは

現在の一般的な Web アプリケーションでは、SessionID と呼ばれる一意でかつ推測困難な値を Web ブラウザ上に保持(大抵はクッキー情報として保持)し、Web ブラウザから渡される SessionID の値を確認することで HTTP セッションの管理を行っている。
この SessionID を他者が利用することができれば、他者の HTTP セッションになりすますことができてしまう。通常、SessionID は Web アプリケーションによって推測困難な値が作成されるため、Web アプリケーションを利用している他者の SessionID を推測することは困難である。

3. Session Hijack (HTTPセッションに対しての攻撃)

Web アプリケーション上の他者の HTTP セッションを乗っ取ることが可能であれば、他者として成りすますことが可能になり、不正行為者が犠牲者としてふるまったり、犠牲者が入力した情報を搾取したり、さまざまなセキュリティ侵害行為が可能となる。
そして、"Web アプリケーション上の他者の HTTP セッションを乗っ取ること" とは、犠牲者の SessionID に対して攻撃(推測、固定化など)するのとほぼ同義になる。

Web アプリケーションの HTTP セッション管理機構に対しての攻撃は、以下が考えられる。

Copyright 2010 NTT Communications Corporation

- 犠牲者のSessionIDを盗み出す[1]
  - 通信の盗聴 (通信経路上のルータの掌握、ARP Poisoning 攻撃やリピータハブの盗聴など)
  - HTTP over SSL に対する MITM 攻撃 (主に SSLv2)
  - HTTP over SSL 上のクッキー情報(SessionID 含)を HTTP 通信(平文通信)にて盗聴する (主にクッキー情報の Secure 属性)
  - XSS(クロスサイト・スクリプティング)攻撃[2]
- 犠牲者の SessionID を推測する
  - Web アプリケーションの SessionID が推測可能であるという脆弱性を使う
  - Web アプリケーションの SessionID に対して推測攻撃を行う
    - バースデーアタックによる攻撃
    - 通常の総当り攻撃
    - 辞書攻撃
    - RainbowCrack[3]
- 不正行為者には既知の SessionID を犠牲者に使わせる(Session Fixation)
  - 有効な SessionID を収集する方法
    - クエリ文字列上の SessionID に対して
      - ✓ショルダーハック
      - ✓Referer ヘッダによる漏洩
      - ✓利用者の操作ミスによる漏洩
    - 連続してアクセスすることで取得
    - Session Adoption 問題
  - 不正行為者には既知の SessionID を犠牲者に使わせる方法
    - クエリ文字列上の SessionID
    - Cookie Monster
- 不正行為者自身の HTTP セッションを犠牲者に使わせる(Session Poisoning)
  - 犠牲者が使うことで、犠牲者の HTTP セッションに変化する(Session Morfing/Session Modification)。
  - 犠牲者が使っても不正行為者自身の HTTP セッションのままで、犠牲者が入力した情報は不正行為者が入力した情報として Web アプリケーションに登録される(Session Surrogate)。

4. Session Fixation攻撃

「2 SessionIDによるHTTPセッション管理とは」で説明したとおり、SessionIDは推測困難な値である。
そこで、攻撃する側が “成りすまされる利用者(犠牲者)” の SessionID を推測するのではなく、不正行為者には既知のセッション ID を犠牲者に使わせようという攻撃手法が「Session Fixation 攻撃」と呼ばれるものである。

---

[1] どうせ盗聴するのであれば、SessionID を盗難するよりも Credential 情報を盗難した方が、効率的かもしれない
[2] クッキー情報に httponly 属性が有効な場合は、XST(クロスサイト・トレーシング)攻撃も考えられる
[3] SessionID 用の RainbowCrack については聞いたことがないが念のため追加してみた

Copyright 2010 NTT Communications Corporation

この場合、不正行為者にとっては犠牲者の SessionID は既知となるため、推測する必要がない。よって、犠牲者がその SessionID を再利用してくれれば、そのまま犠牲者の HTTP セッションを乗っ取ることが可能となる。

Session Fixation 攻撃は、さらに二つの段階に分解することができる。
1. 有効な SessionID を収集する
2. 有効な SessionID を犠牲者に使わせる

後述となるが、Session Adoption 問題は、この二つの段階の「有効な SessionID を収集する」側と関連する問題であるので、本文書ではもう一つの「有効な SessionID を犠牲者に使わせる」というのは概要程度にとどめ深く検討することはしていない。

4.1. 有効なSessionIDを収集する

Session Fixation 攻撃を行うために、不正行為者は犠牲者に配布するための有効な SessionID を収集しなければならない。
一般的な Web アプリケーション・フレームワーク(ASP.NET や ASP,JSP/JavaServlet,PHP など)では、自動的に SessionID を送出する機能がある。
それらの Web アプリケーションに、SessionID を持たずにアクセスすれば、Web アプリケーション・フレームワーク上にセッション・オブジェクトが生成され、それに紐付く SessionID が送出される。
または、クエリ文字列上でSessionIDを保持している場合は、"画面の覗き見(ショルダーハック)" によって収集することも可能性として考えられる[4]。
さらに、クエリ文字列上で SessionID を保持している場合、外部サイトへのリンクなどによって、"Referer" というリンク元の URL(クエリ文字列を含)として外部サイトのサーバのログに書き出され、それを集めるという可能性も考えられる。
この他に、“収集する” という意味とは異なるが、クライアント(Web ブラウザ)が決定した任意の値が SessionID として有効になる場合もある。このような場合は、有効な SessionID を集める必要がない。なぜなら、任意の値を SessionID として用いることができるからだ。このようにクライアントが決定した任意の値が SessionID として有効になる問題が「Session Adoption」問題と呼ばれている。

有効な SessionID を収集する主な方法は、以下の方法が可能性として考えられる
- クエリ文字列上に保持されている場合、
  - ショルダーハックによる "覗き見"
  - "Referer"というリンク元 URL として外部サイト(不正行為者が管理者の外部の Web サイト)のログに漏洩
  - 利用者の操作ミスによる漏洩
- Web アプリケーション・フレームワークで、HTTP セッションを自動生成させている場合は、SessionID がない状態でアクセスすることで、有効な SessionID を集めることができる
- 任意の値の SessionID が有効な場合は、有効な SessionID を収集する必要がない (Session Adoption 問題)

ここで、本文書のテーマのである「Session Adoption」問題が登場した。この問題については、次章(「5 Session Adoption 問題」)以降でさらに詳しく検討する。

[4] *長い SessionID を瞬間的に暗記できるのかどうかという別の問題があるかもしれない*

Copyright 2010 NTT Communications Corporation

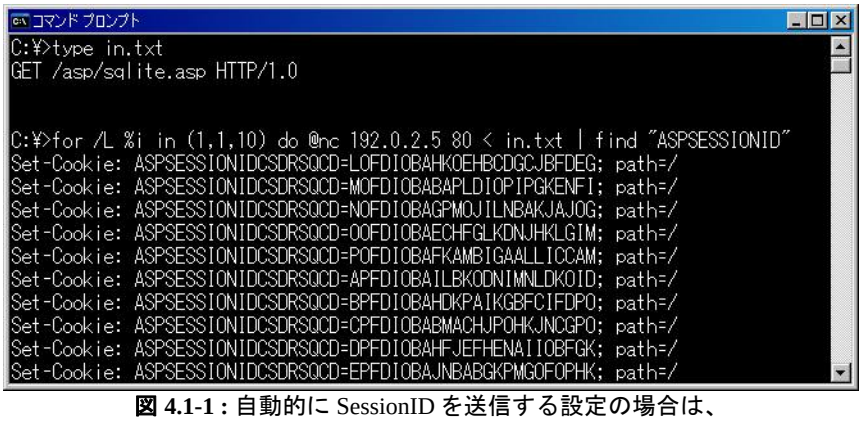


図 4.1-1：自動的に SessionID を送信する設定の場合は、有効な SessionID を収集することは容易である

4.2. 有効なSessionIDを犠牲者に使わせる

主に二つの方法がある。

- クエリ文字列上に保持した SessionID が有効な場合、URL(URL はクエリ文字列を含む)と共に犠牲者に配布(メール、Web ページのリンクなど)する
- Cookie Monster 問題で配布する

有効な SessionID を犠牲者に使わせる方法については、本文書のテーマではないので、割愛する。

5. Session Adoption 問題

Session Adoption 問題は、Session Fixation 問題の一部というべきものであり、クライアント(Web ブラウザ)が決定した任意の値が SessionID として有効となってしまうという問題である。
つまり、Session Fixation 問題の中の「有効な SessionID を収集する」というテーマに関係する問題である。
SessionID には「一意性」と「推測困難性」が必要であり、特に推測困難性という性質は、SessionID による HTTP セッション管理機能の安全性のよりどころとなっている非常に重要な性質である。
よって、そもそもクライアントがSessionIDを任意の値に決定できる機能は必要ではなく、むしろセッション管理機能の安全性を低下させるものであり[5]、執筆者の感覚としてSession Adoption 問題はWebアプリケーション・フレームワークのセキュリティ問題として修正されるべきであると認識している。

しかしながら、多くの Web アプリケーションでは、HTTP セッションを自動的に開始するように設定されているため、Session Adoption 問題がなくても、有効な SessionID を収集することが可能である。よって Session Adoption 問題が存在しなくても、不正行為者の「有効な SessionID を収集する」という行為を防ぐ手段がないのも事実である。

[5] *クライアントがより推測困難な値を自ら選択してくる可能性はほとんどないだろう*

Copyright 2010 NTT Communications Corporation

つまり、不正行為者の「有効な SessionID を収集する」という行為そのものを防ぐためには、Session Adoption 問題への対策だけではなく、自動開始機能を停止することも必要である。

蛇足ながら、自動開始機能を停止することが困難なケースが多いため、それ故 Session Adoption 問題へ対策が熱心に行われていないかもしれない。

次章(「6 PHPのSession Adoption問題」)から、PHPで書かれたWebアプリケーションに限定した上で、かつ自動開始機能を停止している場合におけるSession Adoption 問題の役割について検討していく。

6. PHPのSession Adoption問題

PHP には、Session Adoption 問題が存在する。
また、PHPにはHTTPセッションを自動的に開始しないような設定が存在する(「10 参考 1」)。
この設定によって、PHP には有効な SessionID を収集されないように設定することが可能である。

以下より、PHP のコーディング方法によっては、Session Adoption 問題を用いて不正行為者には既知の SessionID を犠牲者に使わせるような攻撃(Session Fixation 攻撃)が成立する可能性について、検討する。

6.1. 検証環境

OS：CentOS5.4
Apache httpd Server ver2.2.3
PHP ver5.1.6

Copyright 2010 NTT Communications Corporation

6.2. PHPのSession Adoption 問題について

6.2.1 検証結果(機能の確認)

1. PHPは、HTTPセッションの自動起動を抑止することができる(図 6.2-1)。
2. 図 6.2-1の状態で、図 6.2-2のようなPHPスクリプトへアクセスする(図 6.2-3～図 6.2-6)。
3. 図 6.2-3～図 6.2-4は、正常な場合の挙動についての確認である。
   HTTPセッションのない状態でアクセスし(図 6.2-3)、Webアプリケーション・フレームワーク(PHPエンジン)が生成したSessionIDで再アクセスすることで、セッション・オブジェクトが正常に生成されていることを確認した(図 6.2-4)。
   この機能を使えば、ログイン入力画面ではSessionIDを送出することなく、ユーザ認証した場合だけ、SessionIDを送出する、というような芸当が可能である。
   つまり、対象のWebアプリケーションにアクセスするだけで、既知のSessionIDを大量に収集するという行為に耐性を持つことが可能となる。
4. 図 6.2-5～図 6.2-6は、Session Adoptionの場合の挙動を確認している。
   図 6.2-5で、Webブラウザが任意に生成したSessionIDをWebアプリケーションへ渡している。
   その状態で、その任意の値のSessionIDと紐付いたセッション・オブジェクトがWebアプリケーション・フレームワーク上で生成されているかどうかを確認したのが図 6.2-6である。
   図 6.2-6を見るとおり、正常にセッション・オブジェクトは生成されていることが確認できる。

このように、Session Adoption 問題が、Session Fixation 問題の引き金となる場合が、非常に稀なケースとは思われるが、存在することを確認した。

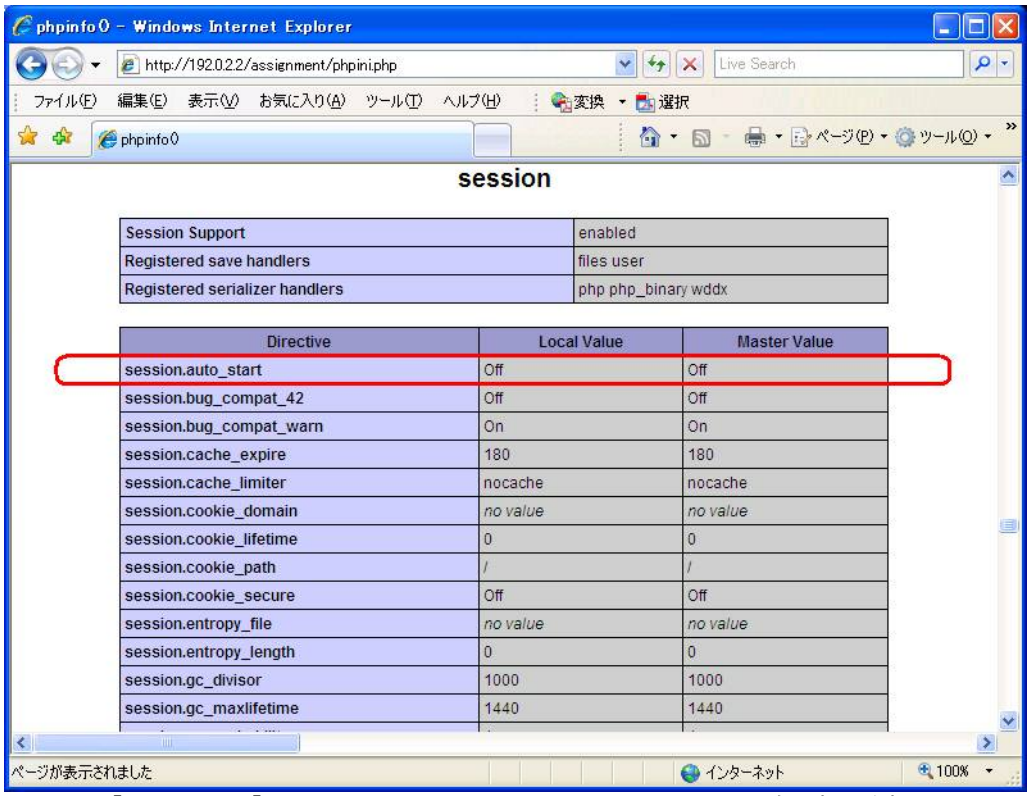


図 6.2-1 : 「php.ini」の「session.auto_start」によって HTTP セッションの自動起動を抑止している

Copyright 2010 NTT Communications Corporation

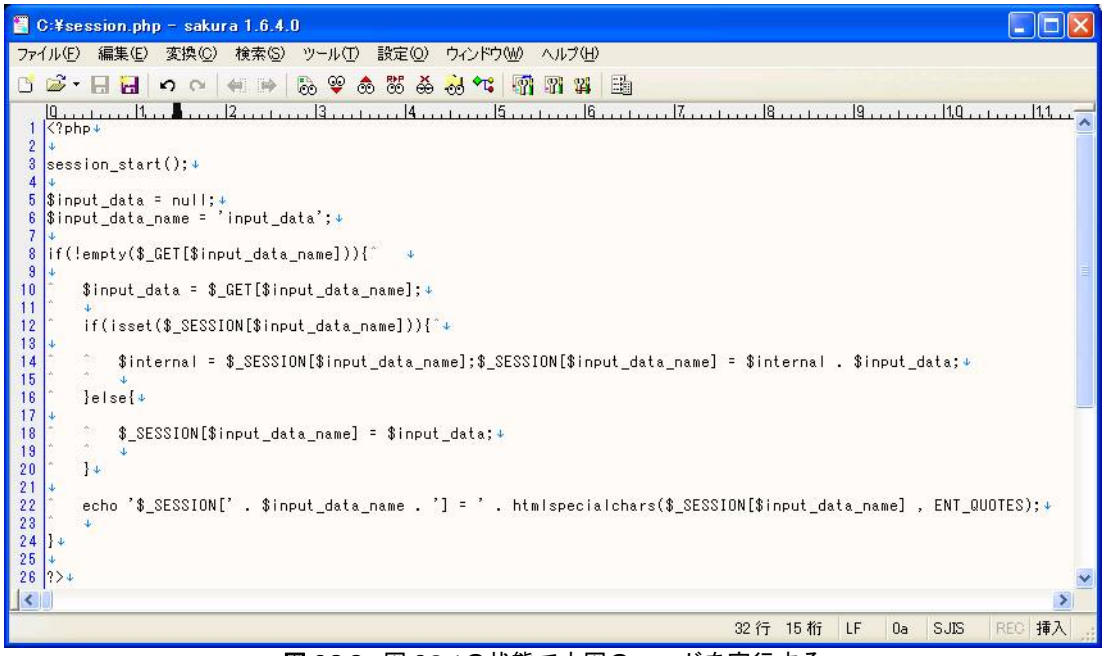


図 6.2-2 : 図 6.2-1の状態で本図のコードを実行する

HTTPセッションを開始する「session_start()」関数が呼ばれている

クエリ文字列「input_data」の値を$_SESSION['input_data']に追記するようにPHPスクリプトである

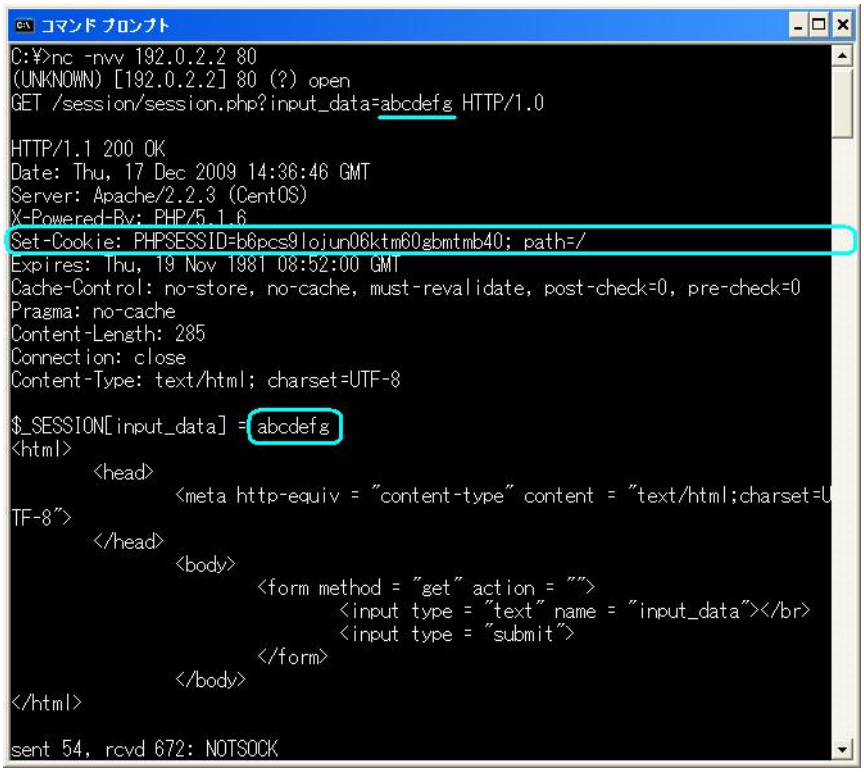


図 6.2-3 : 図 6.2-2に対して、HTTPセッションのない状態でアクセスした結果。

このようにsession_start()関数によって「b6pcs9lojun06ktm60gbmtmb40」というSessionIDの

HTTPセッション・オブジェクトが生成され、セッション変数に「abcdefg」がセットされている

Copyright 2010 NTT Communications Corporation

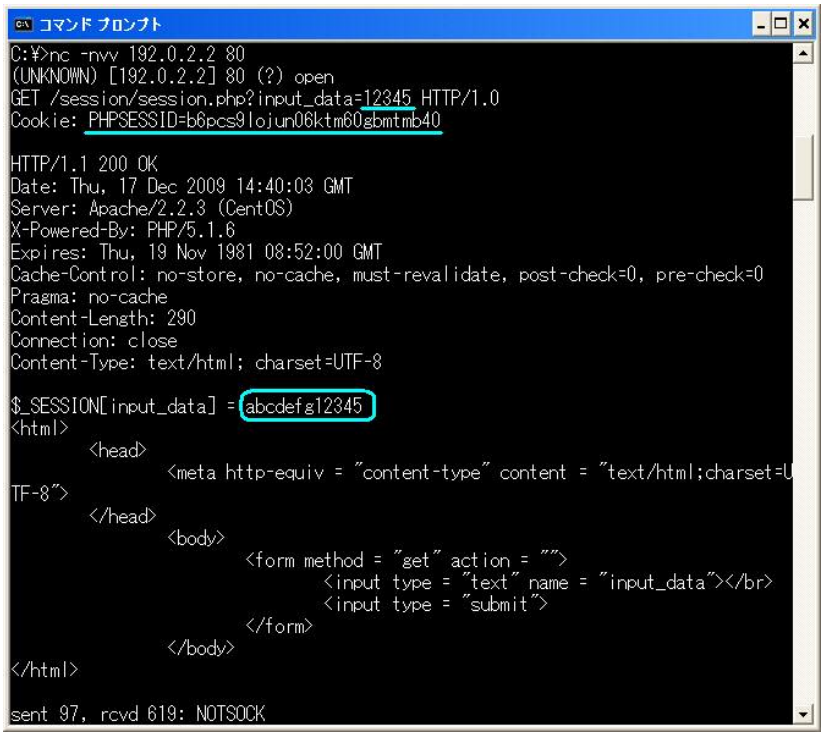


図 6.2-4 : 図 6.2-3後、提示されたSessionIDを使って再度、アクセスした結果。

この結果から、図 6.2-3で作成されたHTTPセッション・オブジェクトが生きており、

図 6.2-3の値(abcdefg)に本図の値(12345)が追記された。

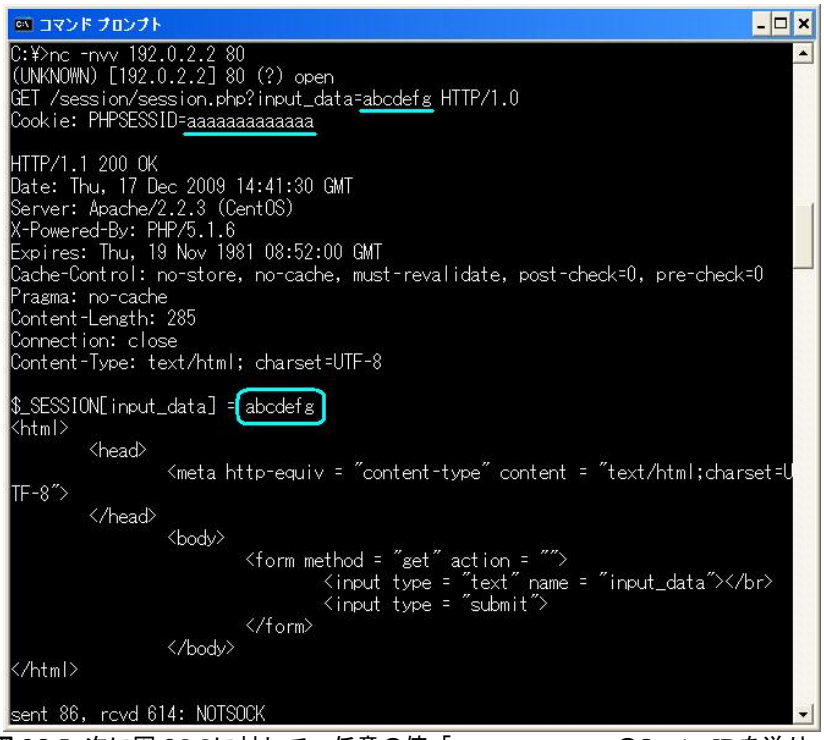


図 6.2-5 :次に図 6.2-2に対して、任意の値「aaaaaaaaaaaa」のSessionIDを送り、

Session Adoptionを行ってみる。

Session Adoptionによって生成されたセッション・オブジェクト上の変数に「abcdefg」がセットされた

Copyright 2010 NTT Communications Corporation

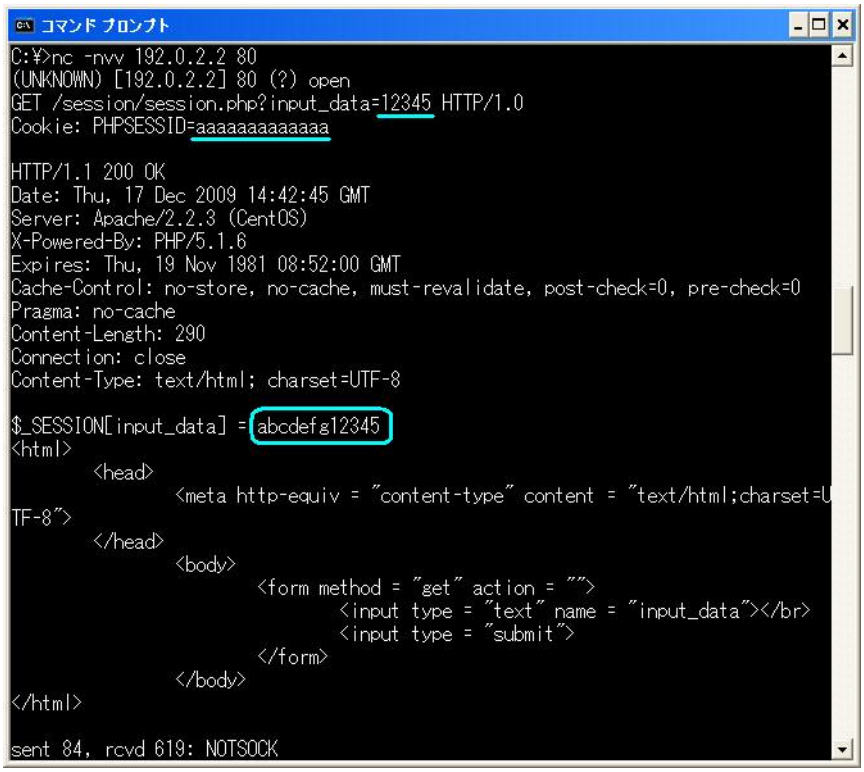


図 6.2-6：図 6.2-5後、再度AdoptionしたセッションIDを使ってアクセスした結果

このようにAdoptionしたSessionIDで紐付いたHTTPセッション・オブジェクトに

「12345」という値が追記されたことからHTTPセッション・オブジェクトが

生きていることが確認できる

6.3. より具体的なPHPのSession Adoption 問題

以下のようなWebアプリケーションを考えてみる(図 6.3-1)。

- 認証情報の入力画面では、SessionID は発行されない
- 認証された利用者のみに SessionID が発行される
- PHP で記述され、session_regenerate_id()関数は使われておらず、session_start()関数のみが用いられている
- SessionID なしで認証画面以外にアクセスした場合、認証画面へリダイレクトされる(この時点では有効な SessionID は発行されない)

つまり、有効な SessionID を収集することができないようにプログラミングされた Web アプリケーションである。
Session Adoption 問題によってのみ、不正行為者には既知の SessionID を犠牲者に使わせることができるような Web アプリケーションのサンプルを作成し、実際に挙動の確認を行った。

ソースコードは、図 6.3-2～図 6.3-4である。
通常の挙動を確認しているのが、図 6.3-5～図 6.3-8である。ユーザ名とパスワードを使ってauth.phpで認証されるまでは、SessionIDは送出されないようになっている。

Copyright 2010 NTT Communications Corporation

また、図 6.3-9～図 6.3-10のように、SessionIDなしでアクセスすることでは有効なSessionIDを収集することはできないようになっている。
このような状態でも、Session Adoption問題がある場合、図 6.3-11～図 6.3-12のように、Session Fixation攻撃を受けてしまう危険性が残ってしまう。

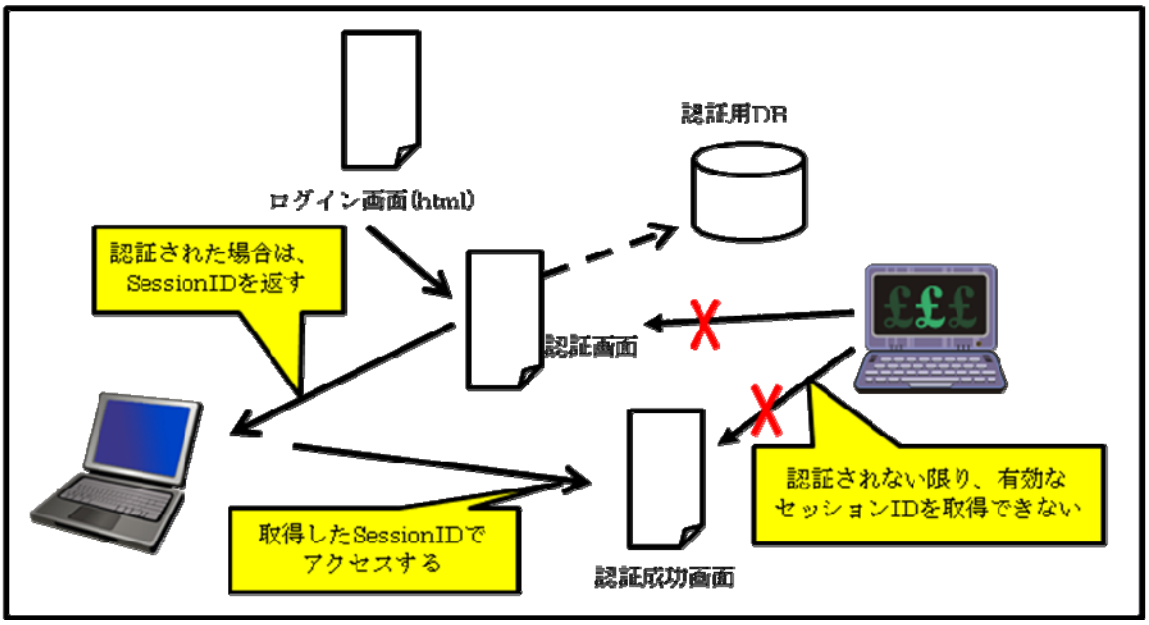


図 6.3-1 : ページの遷移図

これらのページで構成されている Web サイトでは、

認証されない限り有効な SessionID を収集することはできない

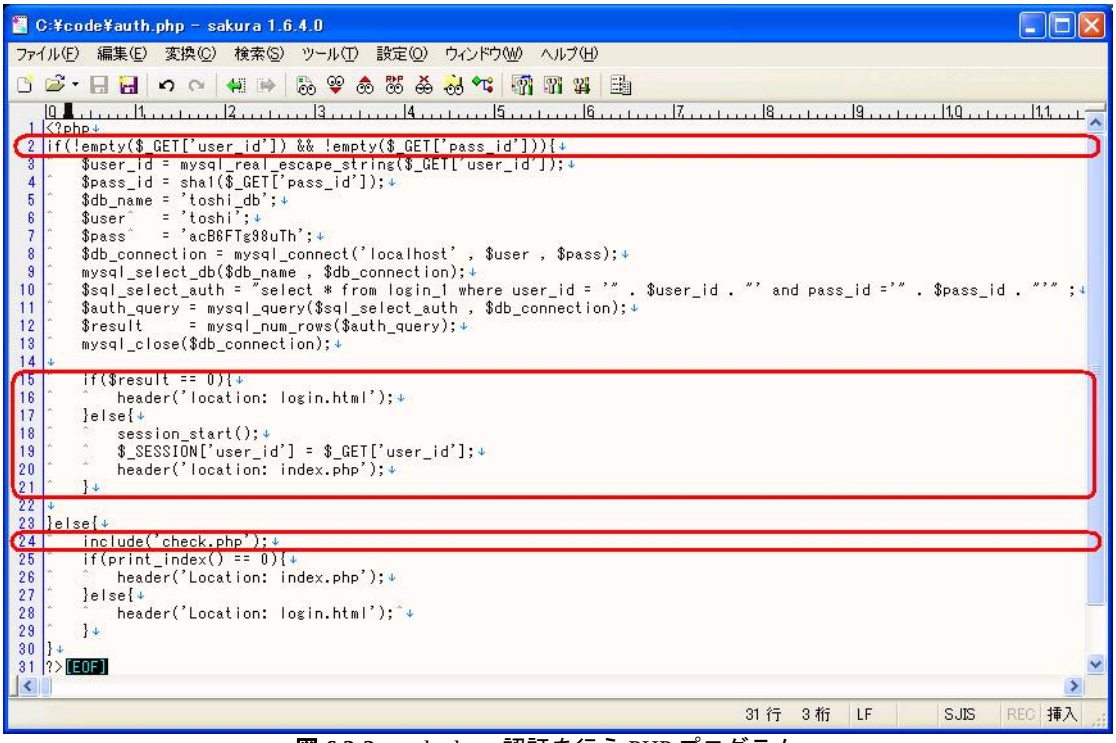


図 6.3-2 : auth.php。認証を行う PHP プログラム。

認証に成功した場合のみ session_start()関数が呼び出され、有効な SessionID が送出される

Copyright 2010 NTT Communications Corporation

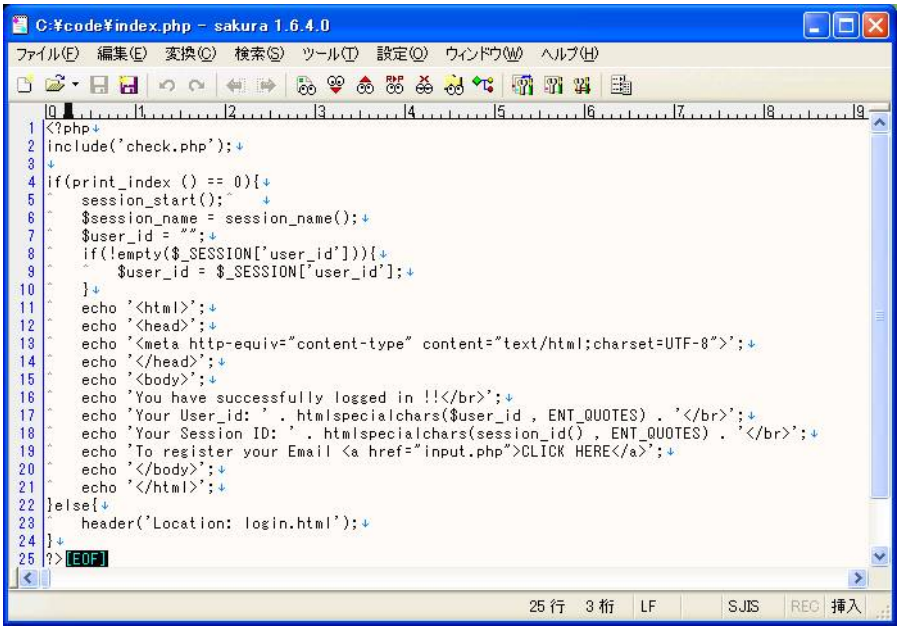


図 6.3-3 : index.php。

認証したユーザ(有効な SessionID を有するユーザ)が訪れる認証ユーザ用トップページ

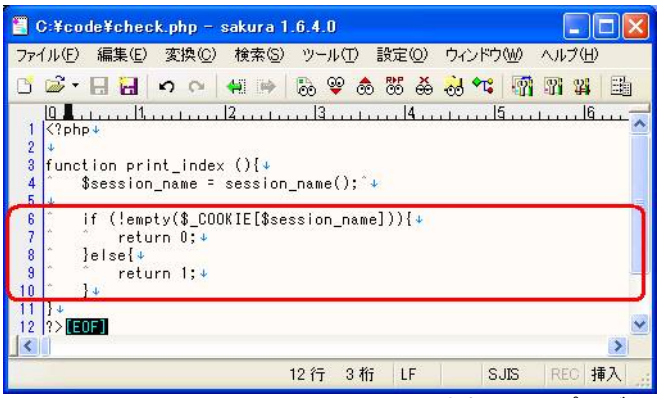


図 6.3-4 : check.php。図 6.3-2や図 6.3-3でIncludeされるPHPプログラムで、SessionIDを保持しているクッキー情報が取得できるかどうかを確認している

Copyright 2010 NTT Communications Corporation

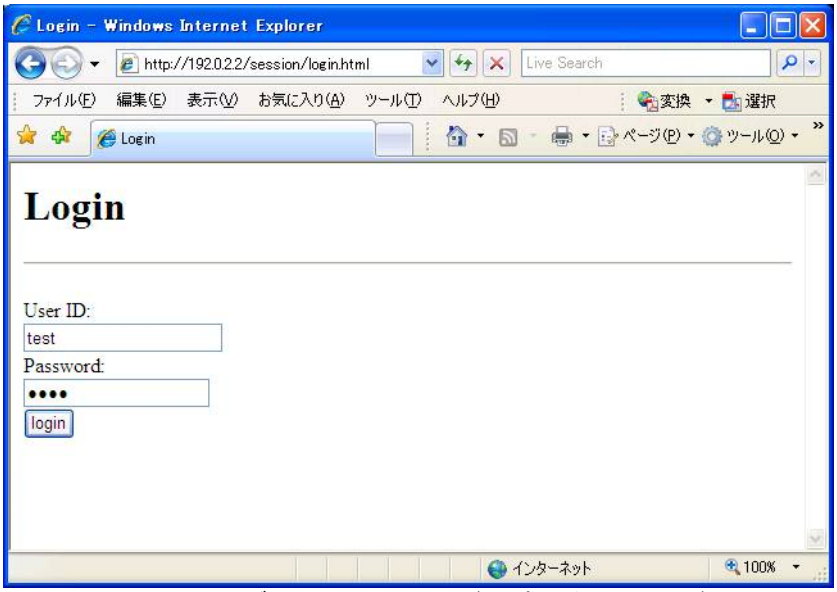


図 6.3-5 : ログイン画面のイメージ(通常動作のイメージ)

ここでユーザ名、パスワードを入力すると・・・

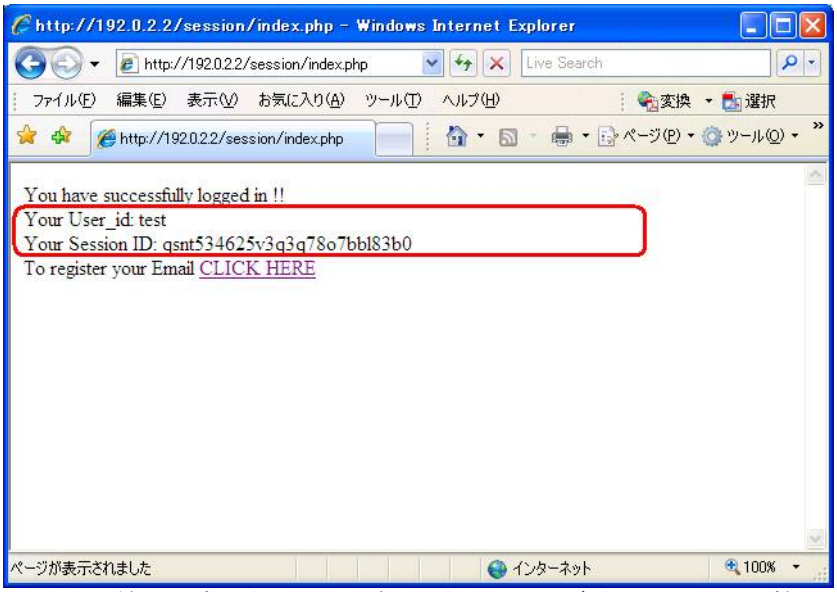


図 6.3-6 : 図 6.3-5の結果(通常動作のイメージ)。認証したユーザだけにSessionIDが払い出される

Copyright 2010 NTT Communications Corporation

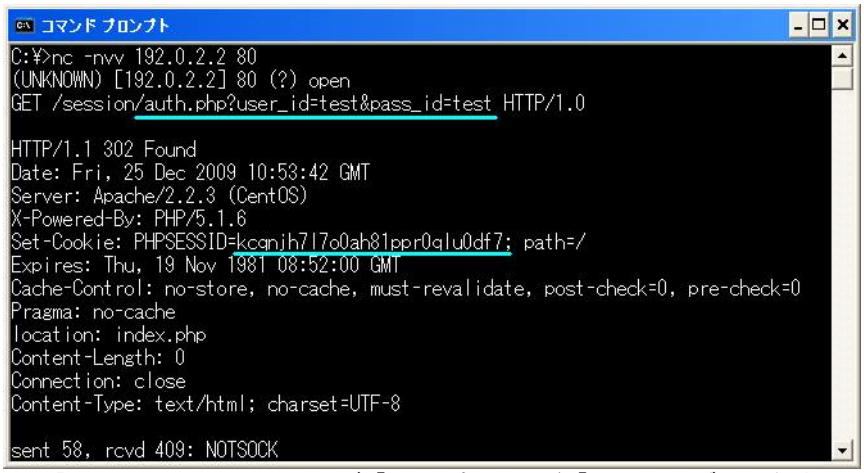


図 6.3-7 : 認証画面(login.html)よりユーザ「test」パスワード「test」でログインを試みた結果。

図 6.3-5の通信データのイメージであるため、認証に成功している。

よって、有効なSessionID「kcqnjh7l7o0ah81ppr0qlu0df7」が送出されている

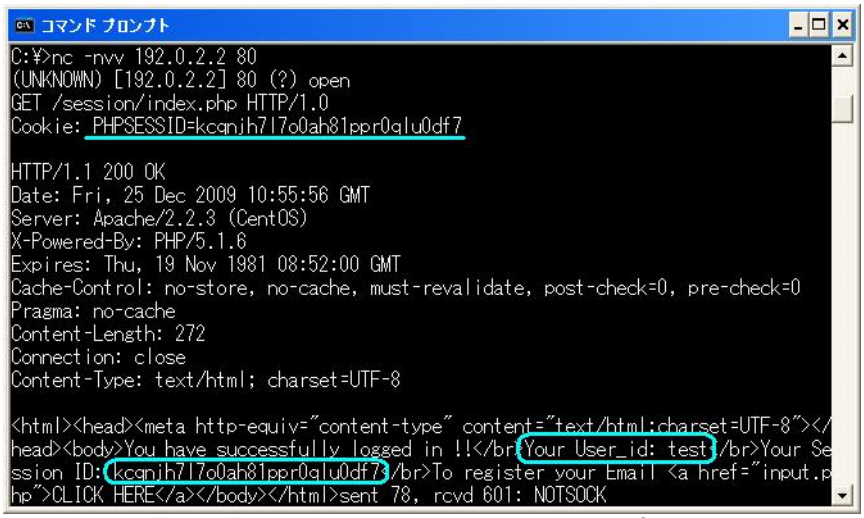


図 6.3-8 : 図 6.3-7後、Webブラウザは、SessionIDを送り、利用者のトップ画面(index.php)へアクセスした。

ユーザID「test」と表示され、セッションIDも正しい値である。

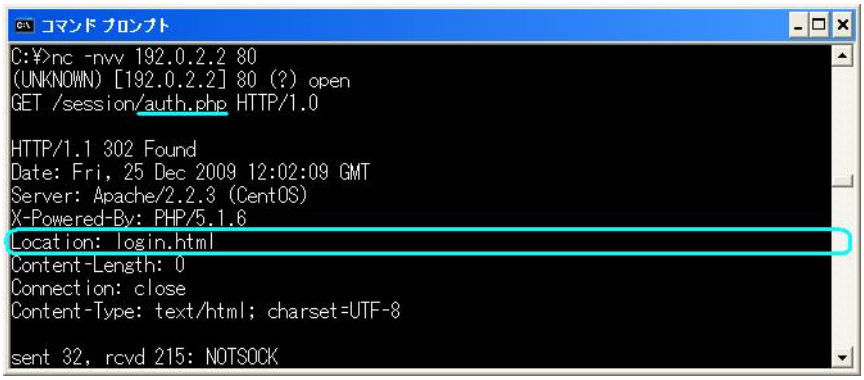


図 6.3-9 : SessionID なし、正しい認証情報なしで auth.php にアクセスしても、

SessionID を取得することできずに login.html へリダイレクトされる。

Copyright 2010 NTT Communications Corporation

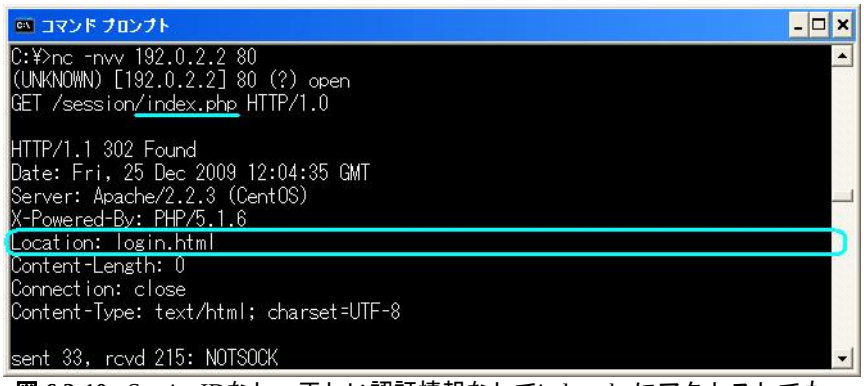


図 6.3-10 : SessionIDなし、正しい認証情報なしでindex.phpにアクセスしても、

SessionIDを取得することできずに login.htmlへリダイレクトされる(図 6.3-9と同様)。

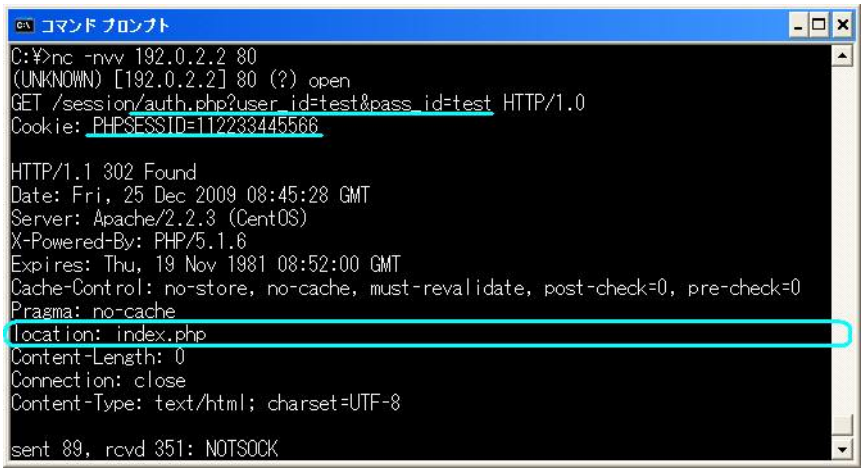


図 6.3-11 : 次は、Session Adoption 問題を使って、

任意の値(112233445566)を SessionID として送り込みつつ認証行う。

犠牲者が入力したユーザ情報により認証され、そのまま index.php へリダイレクトされる

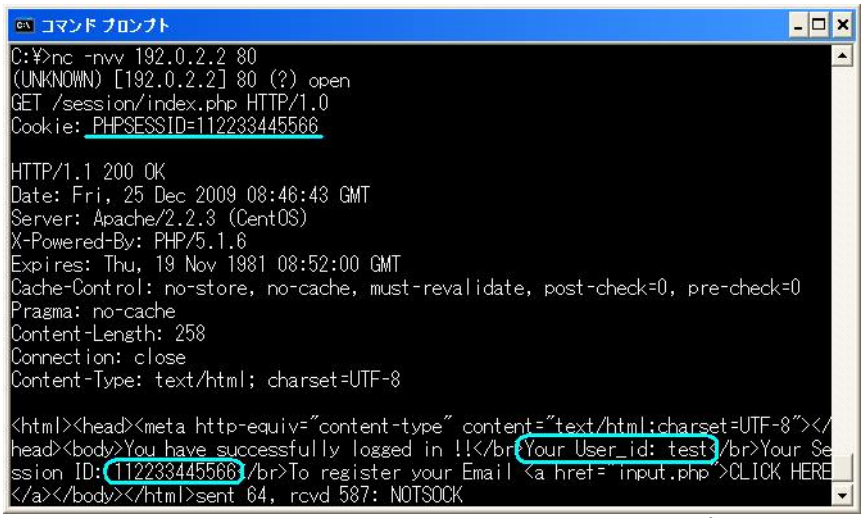


図 6.3-12 : 図 6.3-11後、index.phpへアクセスし、正しく利用者のトップ画面が表示された

このように、図 6.3-9や図 6.3-10のような有効なSessionIDを収集することができない

Webアプリケーションであっても Session Adoption問題が脅威となる場合がある

Copyright 2010 NTT Communications Corporation

6.4. Session Adoption 問題とPlaceFolderのundefined問題

「6.3 より具体的なPHPのSession Adoption 問題」には実は、重要な脆弱性がある。それが図 6.4-1～図 6.4-2である。

図 6.3-2のauth.phpのソースコードを確認してみると、ユーザIDとパスワードが与えられていない場合は(現時点で有効ではなくても)SessionIDがあるかどうかだけで、認証済みか否かを判断している。
よって、図 6.4-1～図 6.4-2のように、認証されていない場合であってもSessionIDを提示するだけで、index.phpへアクセスすることが可能である。
よって、HTTPセッションを自動起動するような設定にしていない場合(図 6.2-1および図 6.3-2～図 6.3-4のようなWebアプリケーション)、HTTPセッションの存在が認証済み利用者からのアクセスであるということではない、という点に注意して認証ロジックを実装する必要である[6]。

さらに、図 6.4-3のようなPHPスクリプトを考えてみる。このスクリプトは、HTTPセッションが有効であれば、ログインしたユーザIDが$_SESSION['user_id']に格納されていると想定し、それを使って本人情報だけを表示させようとしている点がポイントである[7]。
実際、図 6.4-4は、通常のログイン処理(auth.php)を経由してアクセスした結果である。通常のログイン処理によって、$_SESSION['user_id']にログインしたユーザIDがセットされてしまっているため、本人情報のみが表示されている。
一方で、Session Adoptionした場合の図が図 6.4-5である。Session Adoptionしたことで、$_SESSION['user_id']は不定の値であるため、本人情報のみに制限されず全件が表示されていることが確認できる[8]。
このように、HTTP セッションを自動起動しないように設定している場合“HTTP セッションの存在=認証済みとは限らない”という点に注意して認証ロジックを実装することが必要である。

本文書では、セッションIDが存在した場合、$_SESSION['user_id']の値も確認するようにcheck.phpを修正した(図 6.4-6)。
図 6.4-6のように修正することで、図 6.4-1～図 6.4-2のような挙動を防ぐことが可能である(図 6.4-7)。

---

[6] 自動的に HTTP セッション管理機能が起動するように設定している場合、認証前から HTTP セッションが有効になっていることから、このような状況に応じた認証ロジックを実装している開発者がほとんどであると思われるので、あえて「HTTP セッションを自動起動するような設定にしていない場合」という点を強調した

[7] サンプルのコードでは変数がセットされているかどうか判断し、Place Folder が確実に未定義になるようなコードとなっている

[8] 執筆者は、セキュリティ診断作業中において、稀に Session Adoption を引き金にしてこのような情報漏えいとなるようなセキュリティ問題を確認することがあるため、開発者の方は未定義の PlaceFolder がないように注意してほしい

Copyright 2010 NTT Communications Corporation

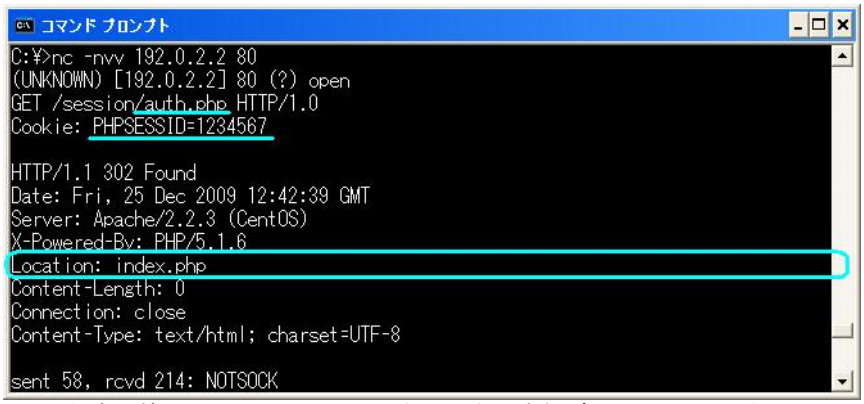


図 6.4-1 :任意の値(1234567)の SessionID を与えさえすれば、auth.php の認証は通過し、index.php へリダイレクトされる

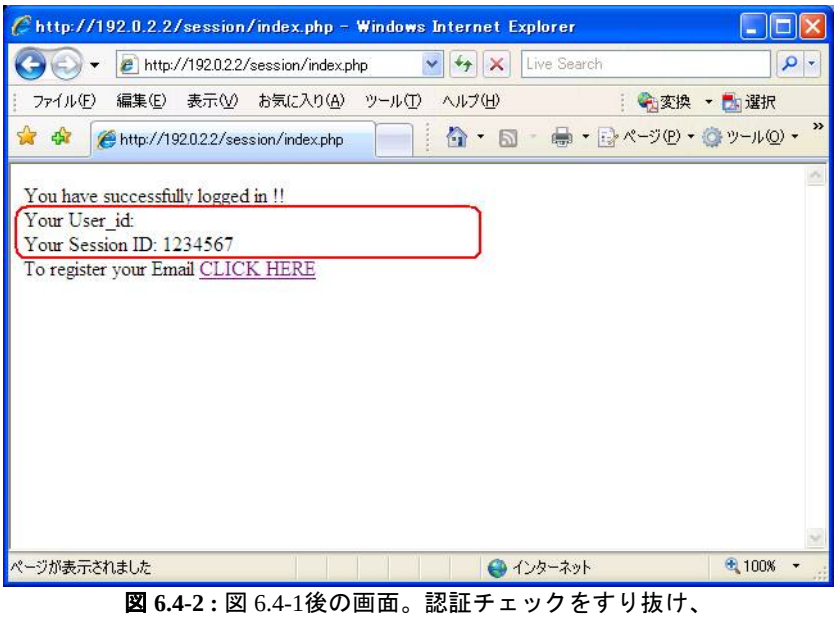


図 6.4-2 : 図 6.4-1後の画面。認証チェックをすり抜け、ユーザIDは不定の状態となっている

Copyright 2010 NTT Communications Corporation

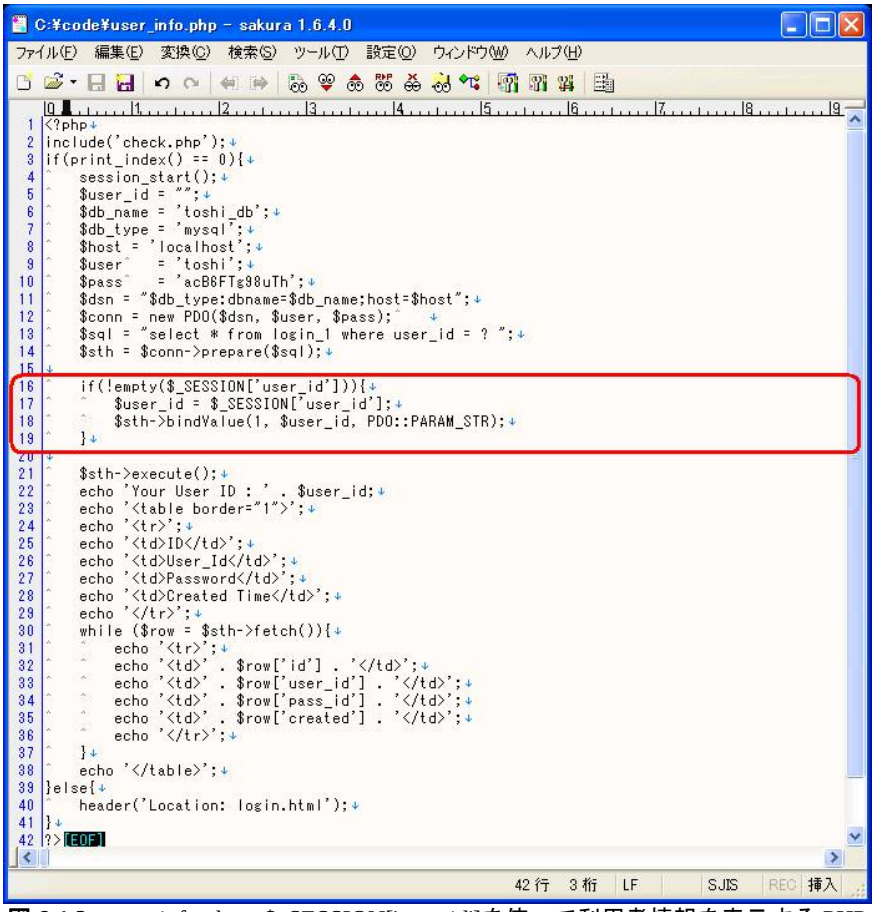


図 6.4-3 : user_info.php。$_SESSION['user_id']を使って利用者情報を表示する PHP

この PHP プログラムの背景には、必ず$_SESSION['user_id']はセットされているはずであり、

よって、本人情報だけが表示されるはずであると想定している点である

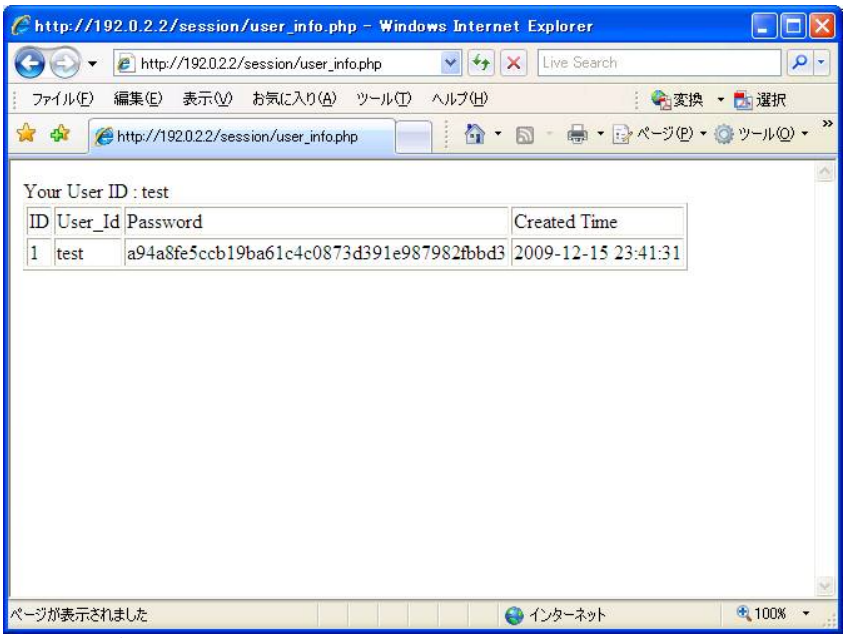


図 6.4-4 : 通常のログイン処理を行っている場合、$_SESSION['user_id']はセットされているため、

図 6.4-3で示したPHPプログラムは、この図のように本人情報しか表示されない。

Copyright 2010 NTT Communications Corporation

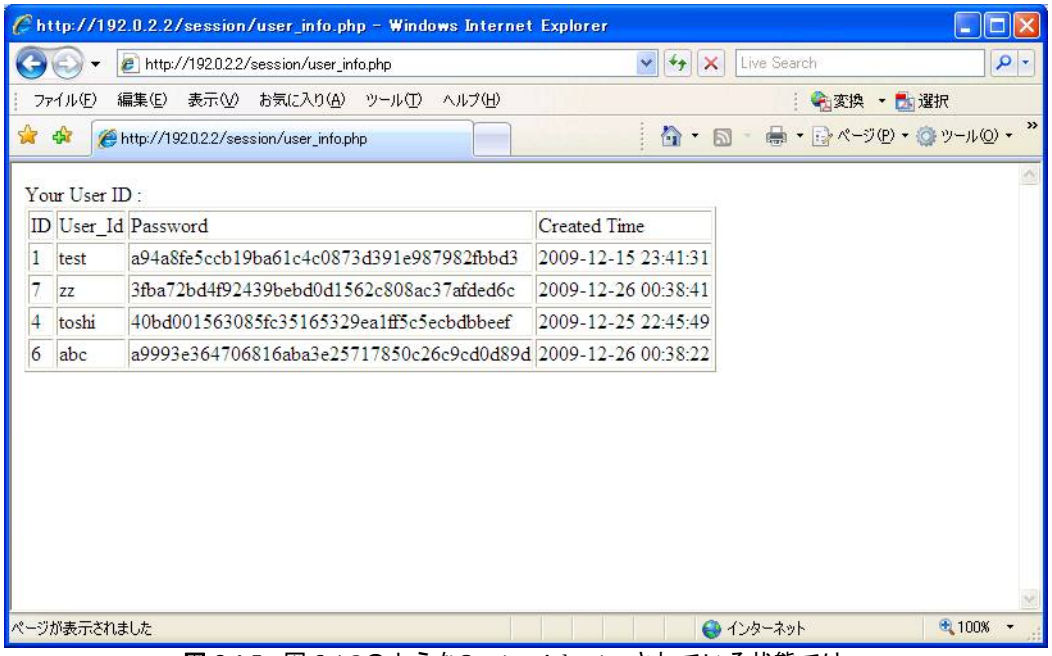


図 6.4-5 : 図 6.4-2のようなSession Adoptionされている状態では、

$_SESSION['user_id']の値がセットされていない。

よって、図 6.4-3のPHPプログラムにアクセスした結果はこのように本人以外の情報も表示されてしまう。

つまり、ユーザIDがセットされている前提が崩れ、全件表示されてしまう事態となっている

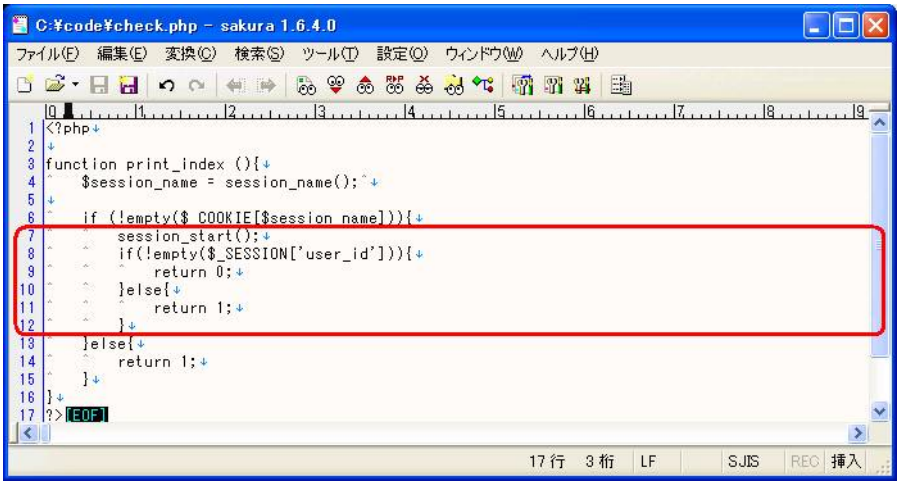


図 6.4-6 : HTTPセッションが有効かどうかの確認を行うcheck.php(図 6.3-4)を修正し、

$_SESSION['user_id']の有無まで確認するようにしたcheck.php

Copyright 2010 NTT Communications Corporation

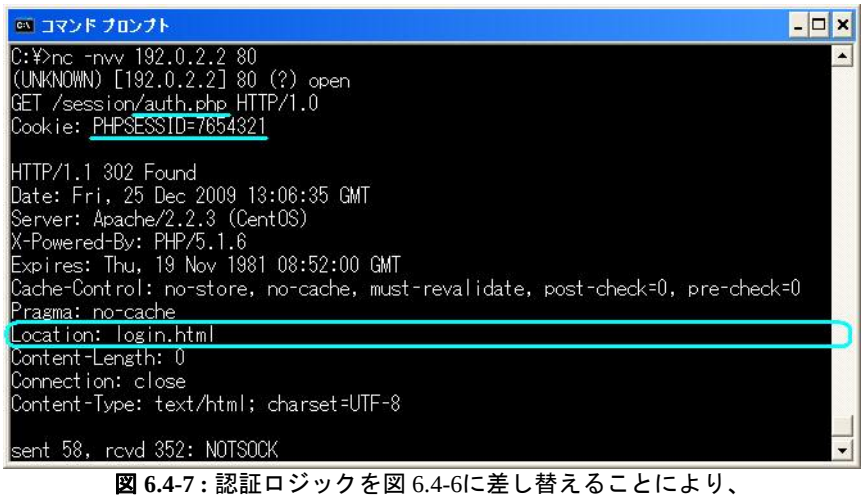


図 6.4-7：認証ロジックを図 6.4-6に差し替えることにより、

図 6.4-1のような認証していない適当なSessionIDを与えても、認証画面へリダイレクトするようになる

6.5. Session Morfing(Session Modification)

「6.3 より具体的なPHPのSession Adoption 問題」で検討した図 6.3-2 のauth.phpと図 6.3-3の index.phpを使い、さらに「6.4 Session Adoption 問題」の図 6.4-6のcheck.phpを使ってさらに、Session Fixation問題を検討していきたい。

「6.3 より具体的なPHPのSession Adoption 問題」で想定していたSessionIDは任意の値であるが、不正行為者のHTTPセッションのSessionIDはどうだろうか。
本章と「6.6 Session Surrogate」は、任意のSessionIDが有効になるSession Adoption問題とは無関係であるが、「6 PHPのSession Adoption問題」からつながるSession Poisoning (Session Modification) の一つとして、検討してみた。

つまり、図 6.5-1～図 6.5-3のように、不正行為者が認証した状態で、このSessionIDを犠牲者に使わせるのである(図 6.5-4～図 6.5-5)。
不運なことに、図 6.3-2 のauth.phpと図 6.3-3のindex.php、図 6.4-6のcheck.phpは、認証後の $_SESSION['user_id']が上書きされる。
よって、図 6.5-1～図 6.5-3後の図 6.5-4のように犠牲者が不正行為者のHTTPセッションのSessionIDを使ってログインしてしまうと、$_SESSION['user_id']が不正行為者の識別情報から、犠牲者の識別情報に書き換わってしまう。
その後に不正行為者にとっては既知である SessionID を使ってアクセスすれば、犠牲者の HTTP セッションを乗っ取ることが可能となる。

しかしながら、「不正行為者が攻撃対象のWebアプリケーションに対してアカウント登録を行い、かつログインすることでSessionIDを取得し、そのSessionIDを犠牲者に使わせる」、という方法は、不正行為者にとってはあまりにも煩雑で非効率[9]であるため、悪用されにくいとは思われるが理論的な不正行為としては考えられるのではないだろうか。

[9] *同時に攻撃を行う犠牲者ごとに、登録済みアカウントと紐付いた SessionID が必要になる*

Copyright 2010 NTT Communications Corporation

対策として考えられるのは、

- 認証情報の上書きを行う場合は、SessionID を再生成する。
- 認証済みであるHTTPセッションに関しては、認証情報(本文書では$_SESSION['user_id'])の上書きを行わない(図 6.5-7～図 6.5-15)。

などが考えられる。

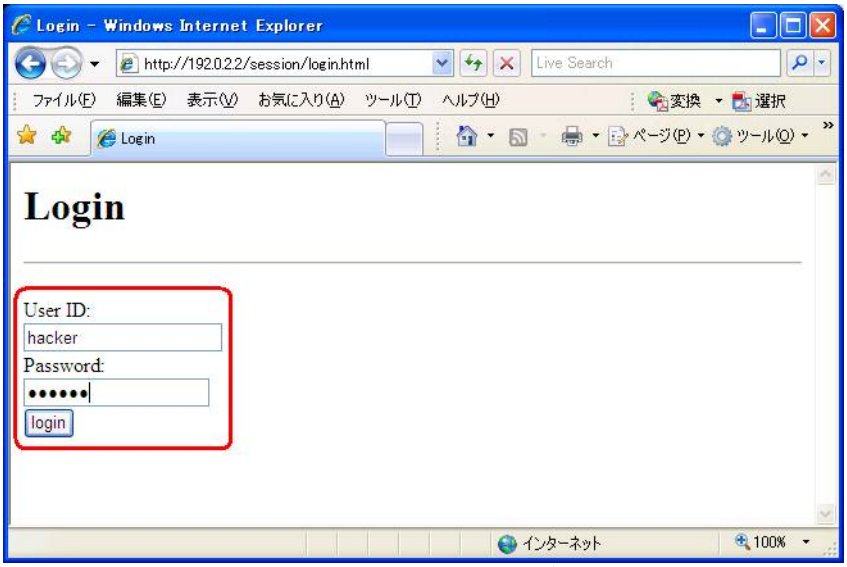


図 6.5-1 : 不正行為者のアカウントでログイン処理を行う

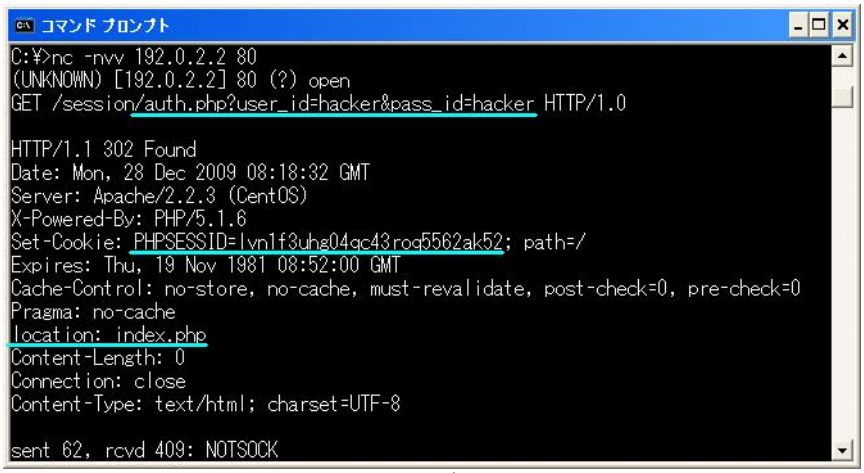


図 6.5-2 : 図 6.5-1後のPHPの画面

不正行為者のHTTPセッションを識別するSessionIDは「lvn1f3uhg04qc43roq5562ak52」である

Copyright 2010 NTT Communications Corporation

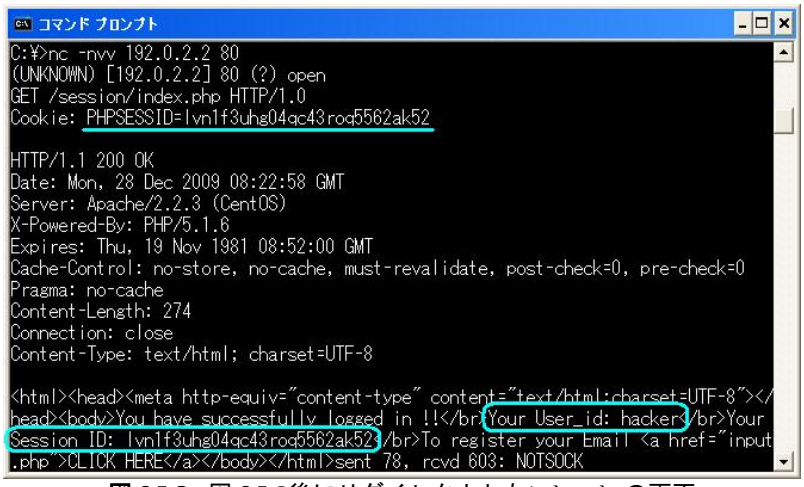


図 6.5-3 : 図 6.5-2後にリダイレクトしたindex.phpの画面

SessionIDは「lvn1f3uhg04qc43roq5562ak52」は不正行為者のHTTPセッション(user_id=hacker)である

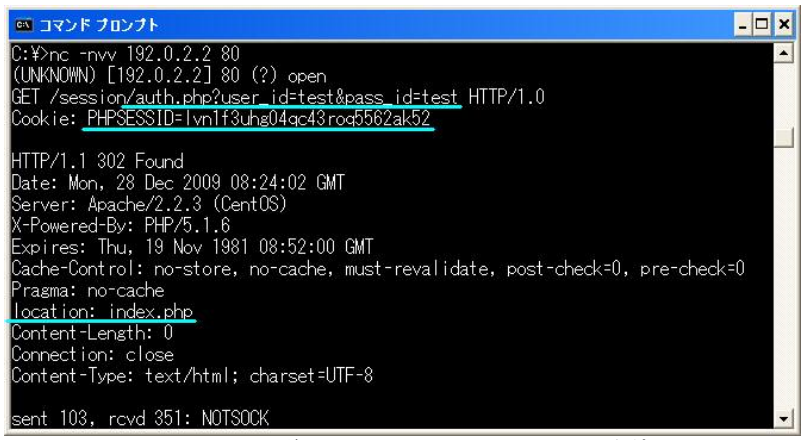


図 6.5-4 : 図 6.5-3後、図 6.5-2と同一のSessionIDを使って

犠牲者がログイン処理を行ってしまうとどうなるだろうか

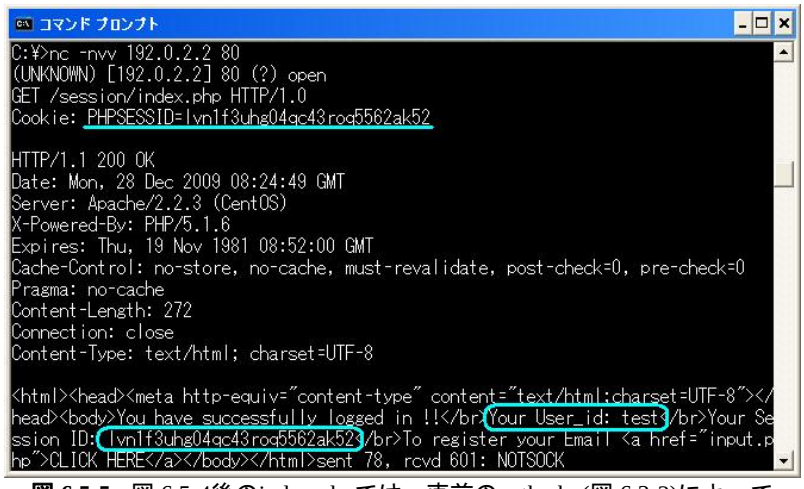


図 6.5-5 : 図 6.5-4後のindex.phpでは、直前のauth.php(図 6.3-2)によって

$_SESSION['user_id']が「hacker」から「test」に上書きされている。

この現象から犠牲者のHTTPセッションを識別するSessionIDは不正行為者には既知となっているため

不正行為者にHTTPセッションを乗っ取られる危険性がある。

Copyright 2010 NTT Communications Corporation

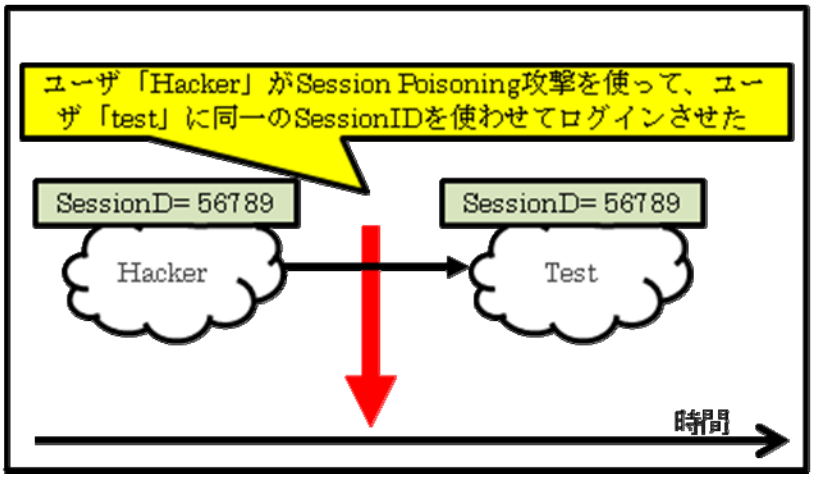
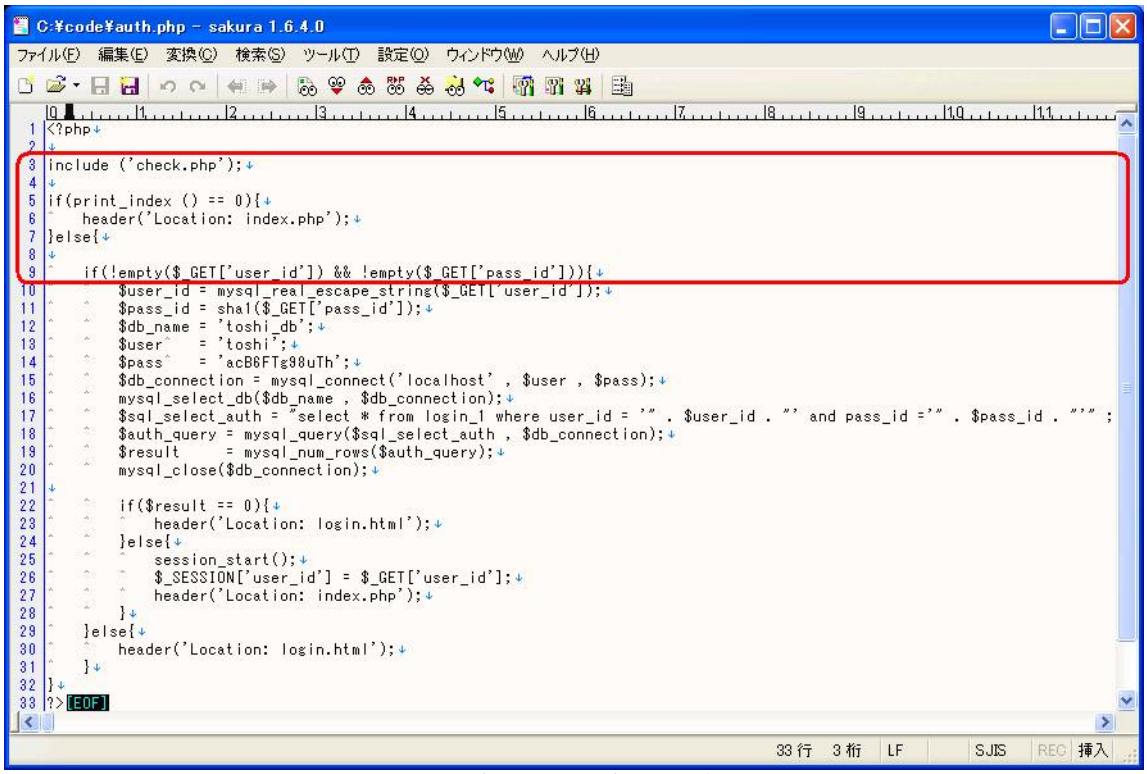


図 6.5-6 : 図 6.5-1～図 6.5-5の流れ。SessionIDは同一のまま、

「hacker」のHTTPセッションが「test」のHTTPセッションへ変化した




```
<?php
include ('check.php');
if(print_index () == 0){
    header('Location: index.php');
}else{
    if(!empty($_GET['user_id']) && !empty($_GET['pass_id'])){
        $user_id = mysql_real_escape_string($_GET['user_id']);
        $pass_id = sha1($_GET['pass_id']);
        $db_name = 'toshi_db';
        $user    = 'toshi';
        $pass    = 'acB6FTg88uTh';
        $db_connection = mysql_connect('localhost' , $user , $pass);
        mysql_select_db($db_name , $db_connection);
        $sql_select_auth = "select * from login_1 where user_id = '" . $user_id . "' and pass_id ='" . $pass_id . "'" ;
        $auth_query = mysql_query($sql_select_auth , $db_connection);
        $result     = mysql_num_rows($auth_query);
        mysql_close($db_connection);

        if($result == 0){
            header('Location: login.html');
        }else{
            session_start();
            $_SESSION['user_id'] = $_GET['user_id'];
            header('Location: index.php');
        }
    }else{
        header('Location: login.html');
    }
}
?>[EOF]
```

図 6.5-7 : 次にauth.phpを図のように、

$_SESSION['user_id']が存在する場合は、認証情報を上書きしないように書き換えた

詳細はcheck.php(図 6.4-6)を参照

Copyright 2010 NTT Communications Corporation

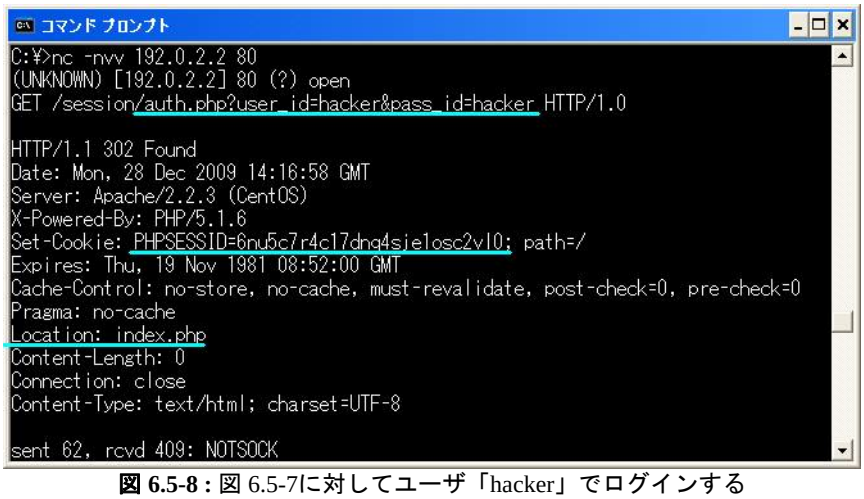


図 6.5-8 : 図 6.5-7に対してユーザ「hacker」でログインする

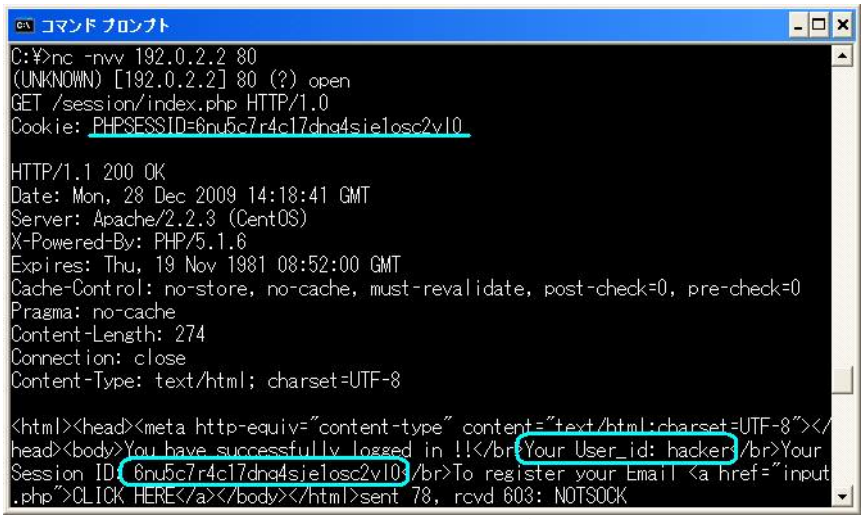


図 6.5-9 : 図 6.5-8後にindex.phpにリダイレクトした結果

「hacker」のSessionIDは「6nu5c7r4c17dnq4sje1osc2vl0」となっている

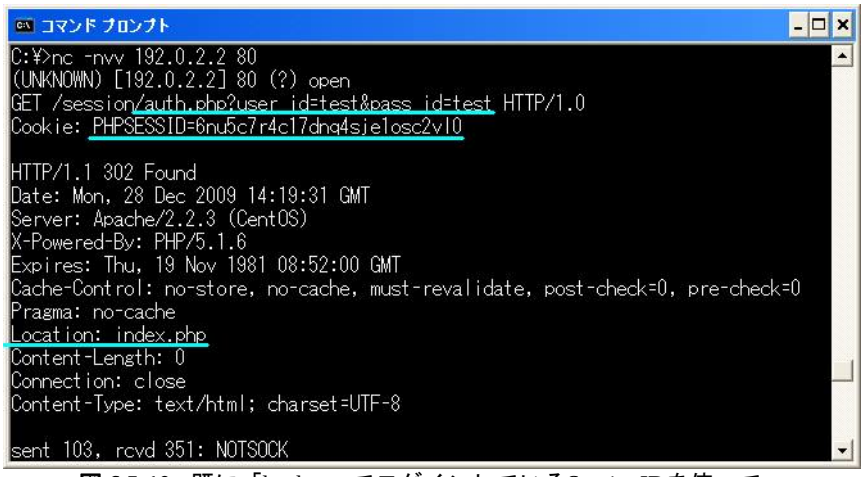


図 6.5-10 : 既に「hacker」でログインしているSessionIDを使って、

図 6.5-7に対してユーザ「test」がログインを試みている

Copyright 2010 NTT Communications Corporation

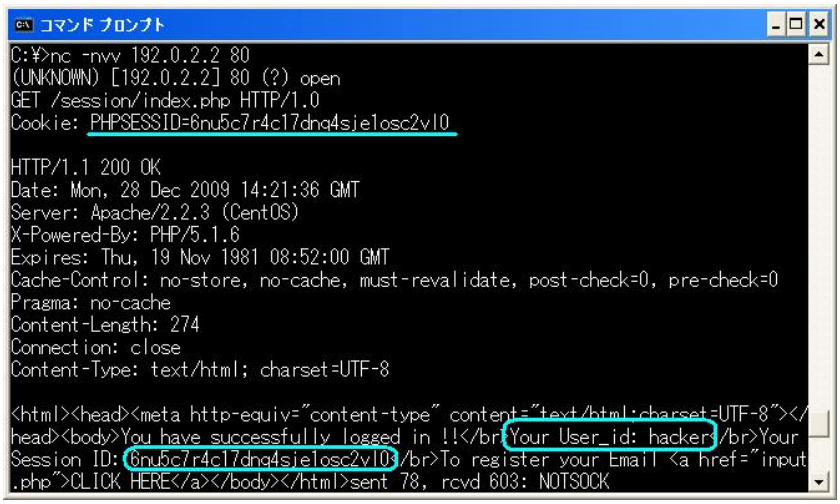


図 6.5-11 : 図 6.5-10の結果。HTTPセッションは「hacker」のままである

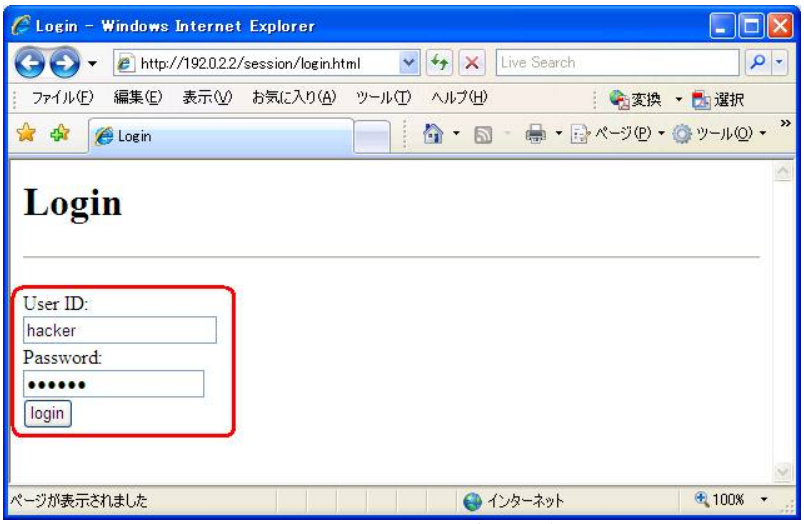


図 6.5-12 : 図 6.5-8～図 6.5-11までをWebブラウザの画面で見てみる。

これは、ユーザ「hacker」でログインする画面

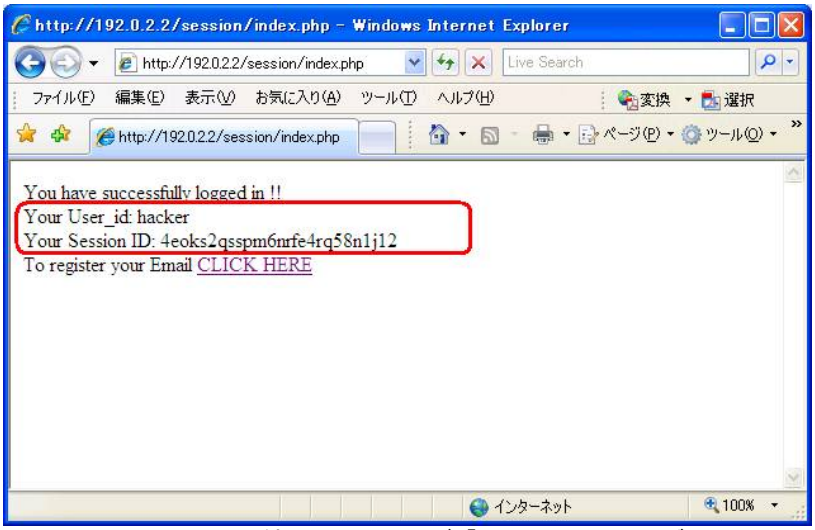


図 6.5-13 : 図 6.5-12の結果画面、ユーザ「hacker」としてログインし、

SessionIDは「4eoks2qsspm6nrfe4rq58n1j12」となっている

Copyright 2010 NTT Communications Corporation

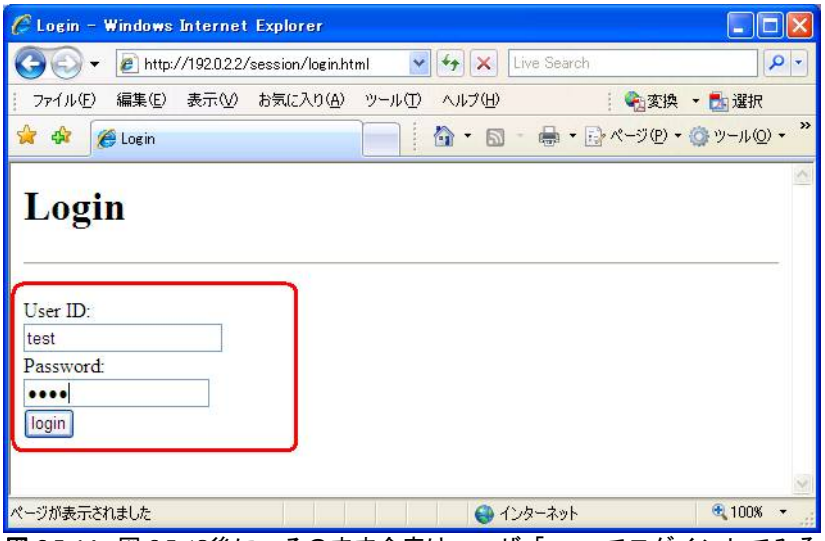


図 6.5-14 : 図 6.5-13後に、そのまま今度はユーザ「test」でログインしてみる

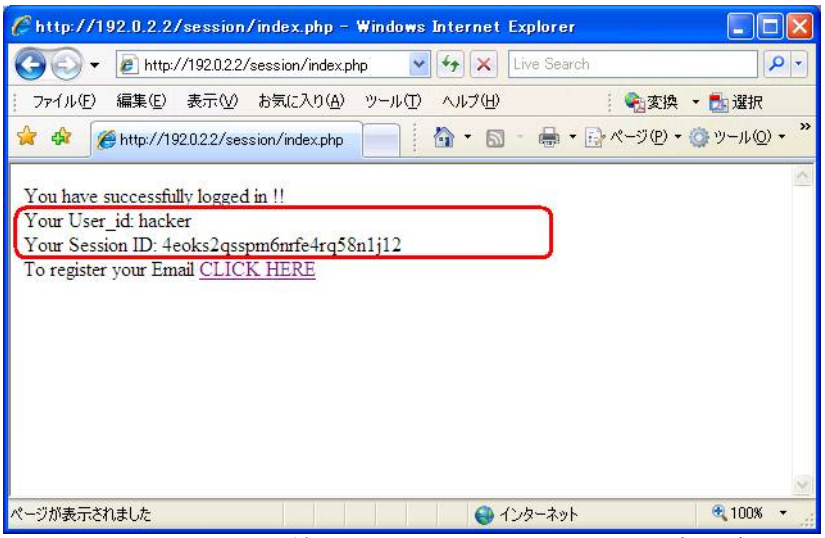


図 6.5-15 : 図 6.5-14の結果。図 6.5-13と同一のSessionIDであるが、
ユーザは「hacker」のままでユーザ「test」へ変化することはなかった

6.6. Session Surrogate (Sessionサロゲート)

「6.5 Session Morfing」では、不正行為者のHTTPセッションが犠牲者のHTTPセッションに変化したが、変化しない場合でも個人情報などの搾取するためのHTTPセッション管理への攻撃は理論的には考えられる。
本章では、この件について検討していく。

不正行為者にとっては「6.5 Session Morfing」よりもさらに条件が厳しくなり、「Webページ上に誰がログインしているか表示されていない」など社会工学的な条件などが必要になる。
さらに、不正行為者にとっては、他者になりすますことができるわけではないため、悪用の効果も限定的である。

Copyright 2010 NTT Communications Corporation

セキュリティ診断を行っていると、このようなユーザ情報を画面に表示していない Web ページが稀にではあるが存在する。
その場合、自分が現在利用している HTTP セッションが、自分のものであるかどうか認識できない状態となってしまう。
このような Web ページでは、利用者の不注意も加わり、他者の HTTP セッションを使っていることに気づかずに、自分自身の個人情報/パスワードなどを入力してしまうという危険性も考えられる。

実際の攻撃可能性は「6.5 Session Morfing」以上に不正行為者にとっては煩雑で非効率であるため、極めて低いと考えられる。

本項に関しての対策は以下が考えられる。

- Web ページ上には、「こんにちは○○さん」や「○○さんがログイン中」のように“誰” の HTTP セッションであるかを明示しておく

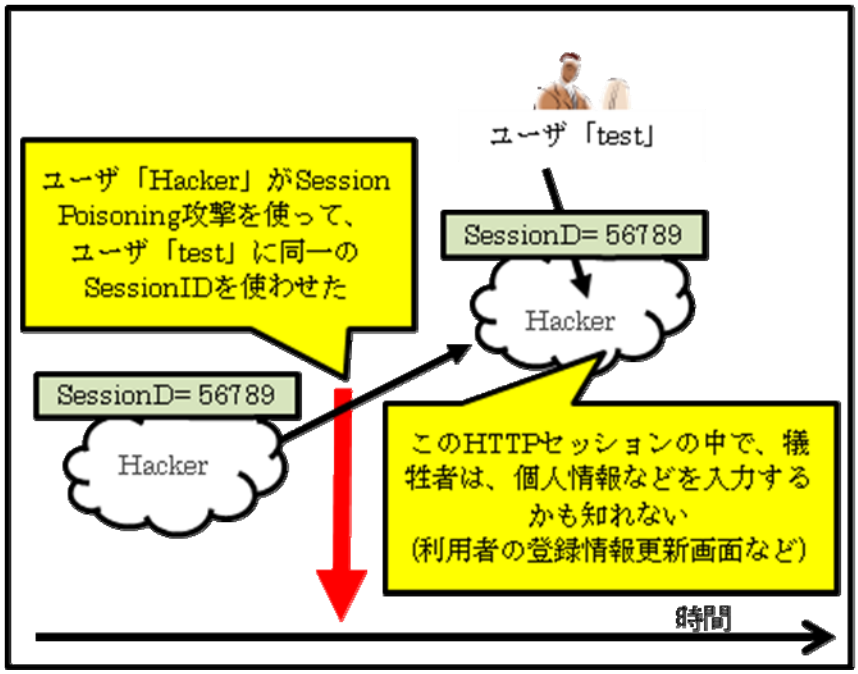


図 6.6-1： 図 6.5-6のように変化しなくても、不正行為者のHTTPセッションを使って、犠牲者は個人情報などの機密情報を入力するかも知れない

7. Session Hijackの現実的な脅威

他者の HTTP セッションを乗っ取るという攻撃(Session Hijack)は、現実的にどの程度の脅威があるのだろうか。
Session Hijack 攻撃の現実的な危険性を検討することで、その一部である Session Fixation 攻撃の現実的な危険性も推定できるのではないだろうか。

他者の HTTP セッションを乗っ取るという攻撃は、結局のところ"他者になりすます" ということに帰着してしまう。

Copyright 2010 NTT Communications Corporation

"他者になりすます" といったセキュリティ上のテーマを考えてみた場合、HTTP セッションを乗っ取る以外にも以下の脅威が存在する。

- 他者のユーザ名/パスワード(Credential 情報)を盗む

こちらの場合と、HTTP セッションを乗っ取る場合との違いについて、考察した。

HTTP セッションの乗っ取り	● HTTP セッションに設けられたタイムアウトの制限がある ● HTTP セッションを維持し続けなければならない
credential 情報の盗難	● 犠牲者がパスワード変更を行わない限り、いつまでも利用可能

また、「Credential 情報の盗難」の代表としてフィッシング詐欺と、Session Poisoning の代表として Session Fixation 攻撃について、具体的な手順を比較してみた。

Session Fixation 攻撃	1. 有効な SessionID を収集 2. 収集した SessionID を犠牲者にばら撒く 3. 犠牲者が罠にかかる 4. 罠にかかった犠牲者の SessionID を特定することで、不正行為者は HTTP セッションの乗っ取りが完成する
フィッシング詐欺	1. フィッシングサイトの開設 2. 犠牲者にフィッシングサイトの URL をばら撒く 3. 犠牲者が罠にかかる 4. 後ほど、犠牲者の Credential 情報を使って、成りすます

段階として、共に 4 段階であるが、Session Fixation攻撃に関しては、これをHTTPセッションのタイムアウト[10]の制限以内[11]に実行しなければならないのに比べ、フィッシング詐欺においては、とくに時間的制限は必要とされない[12]。
このように考えると、Session Poisoning するぐらいなら、フィッシング詐欺などで Credential 情報を収集した方が、不正行為者としてはコスト・パフォーマンスが高いと考えてもよいだろう。つまり Session Poisoning 攻撃自体、PKI 認証(電子証明書)の普及がそれほど進んでいない現時点(2009 年末～2010 年初頭)では危険性が高いセキュリティ上の問題とは言えないと評価してもよいだろう。

8. まとめ{PHPのSession Fixation対策(Session Adoption対策)}

「6 PHPのSession Adoption問題」で検証したように、自動でHTTPセッションを開始しないように構成されたPHPで書かれたWebアプリケーションでは、プログラミングの方法に問題があるとしても、Session Adoption問題が存在するためにセキュリティ上の脅威になるような場面は確認された。

また、有効なSessionIDを収集するための事前準備も必要ない[13]という点でも、Session Adoption問題をセキュリティ上の問題と捉えるべきではないだろうか。

[10] 実際には、3 で犠牲者が罠にかかるときに Web アプリケーションにアクセスし、タイムアウトはリセットされるので、タイムアウトの 2 倍の時間の制限という表現がより適切である
[11] ハートビートを行うなどで、タイムアウトを延長させることができる場合もあるので、困難なハードルではないかも知れない
[12] "フィッシングサイトである" とアンチウィルスソフトベンダーに認定される場合を考えると、時間的制限がないわけではない

Copyright 2010 NTT Communications Corporation

しかしながら、「非常に大きな影響を与える脆弱性」とは執筆者自身も考えてはいない。
Session Adoption問題は、SessionFixation問題の一部と考えることができると、筆者は考えている。よって、「高」「中」「低」という大まかな三段階評価でイメージすれば、「低」(場合によっては「中」[14])程度の危険性を有するセキュリティ上の問題ではないだろうか、というのが執筆者の個人的な感想である[15]。

実際の攻撃可能性について吟味してみると、Session Fixation 問題によって、盗難できるのは、所詮は HTTP セッションだけである。フィッシングサイトによって利用者のアカウントを盗難する方が、手間も少なくかつ利益も大きいのではないだろうかと考えると、不正行為者によって実際に悪用されにくい脆弱性ではないだろうか。

さて、PHP の場合、Session Adoption 問題が修正される気配はない。よって、このようなセキュリティ上の問題となる Web アプリケーションを作らないように、Web プログラマが注意深くプログラムを作成する必要がある。

PHP の場合の Web プログラマ側での対策は、session_regenerate_id()関数を使い、SessionID を再生成すること、つまり不正行為者が犠牲者に使わせた任意の SessionID を破棄して、Web アプリケーションが推測困難な ID に再生成することである。これによって Session Adoption 問題を解決することができる。
より一般的な(Session Adoption 問題を含む)Session Fixation 問題へ対策は以下のような項目などが考えられる。

- 認証後(ログイン処理が代表であるが「自動ログイン機能」がある場合は、ログイン機能に限定されない)に、Webアプリケーション側でSessionIDを再生成する[16]
- セキュリティ重視の利用者のために、ログオフ処理を用意し速やかに有効な HTTP セッションを破棄させる
- HTTP セッションの有効時間を適切に短いものにする
- 同時ログインを禁止し、再ログインした場合、以前の HTTP セッションを破棄する(ユーザ 1 人に 1 つの HTTP セッションの原則)

しかしながら、PHP のような Web アプリケーション・フレームワークが、クライアント(Web ブラウザ)が設定した任意の値を SessionID として有効にできるということは、Web プログラマによる注意深いセキュアプログラミングで解決可能とはいえ、Web アプリケーション・フレームワークのセキュリティ上の問題ではないだろうか。

蛇足ではあるが、PHP ver5.1.0 以降のsession_regenerate_id()関数では、第一引数が定義され、古いセッション・オブジェクトを破棄することも可能となった(図 6.6-1)。
今後は、session_regenerate_id()関数を使う際に古いSessionIDでアクセスさせる必要がない場合は、第一引数に「true」を与え、古いセッション・オブジェクトを破棄するようにプログラミングすることを推奨する(「10**参考**」の11)。

---

[13]監視センターを有する場合は、この準備段階で攻撃の予兆を捉えることができるかもしれない
[14] 人気の Web サイトであることや、自動ログインが有効であること、クエリ文字列上の SessionID が有効であることなどが危険性を高める条件となるだろう
[15] Session Fixation と Session Adoption の関係は、XSS 問題の付属問題としての XST 問題の関係を髣髴とさせるのは執筆者だけだろうか
[16] クッキー情報に SessionID を保持している場合は格段に問題とならないが、クエリ文字列や Hidden フィールドに保持している場合、頻繁に SessionID を書き換えてしまうと HTTP セッションが引き継がれないなどの弊害が発生するかもしれない

Copyright 2010 NTT Communications Corporation

もう一つ蛇足ながら、フレームワークの機能として、Session再生成機能がない場合などは、
Session変数とクッキー情報上に推測困難な値を保持しておき、適時その値を再生成することで、
再生成機能と代わりとすることも可能である(図 6.6-2)。

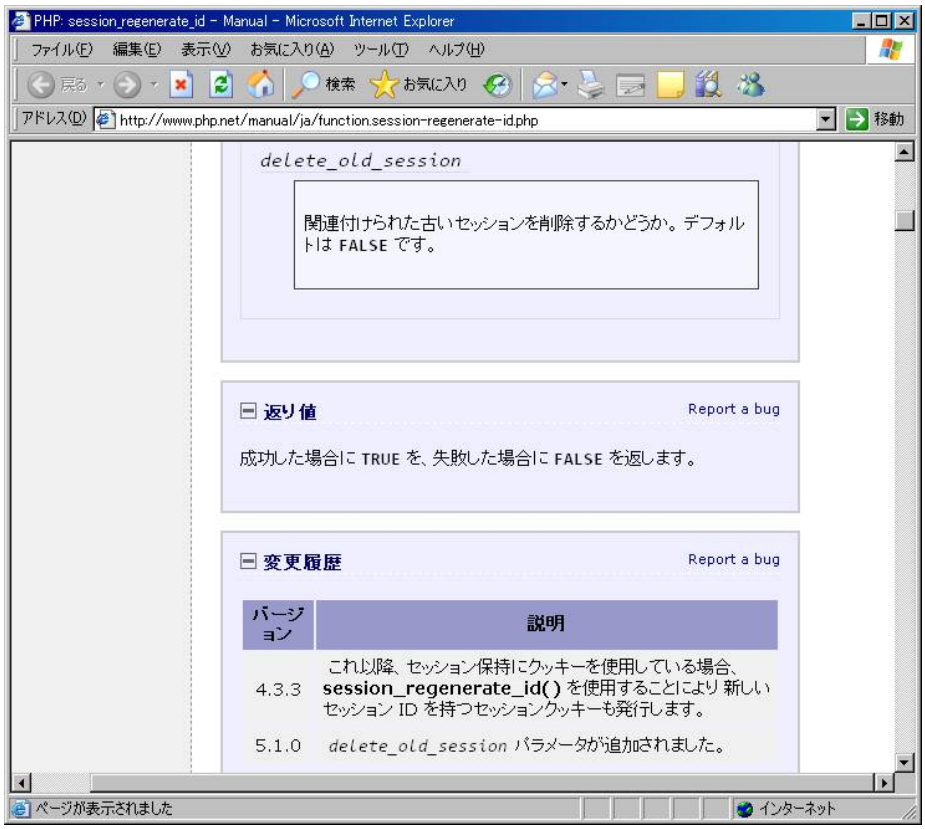


図 6.6-1 : http://www.php.net/manual/ja/function.session-regenerate-id.php

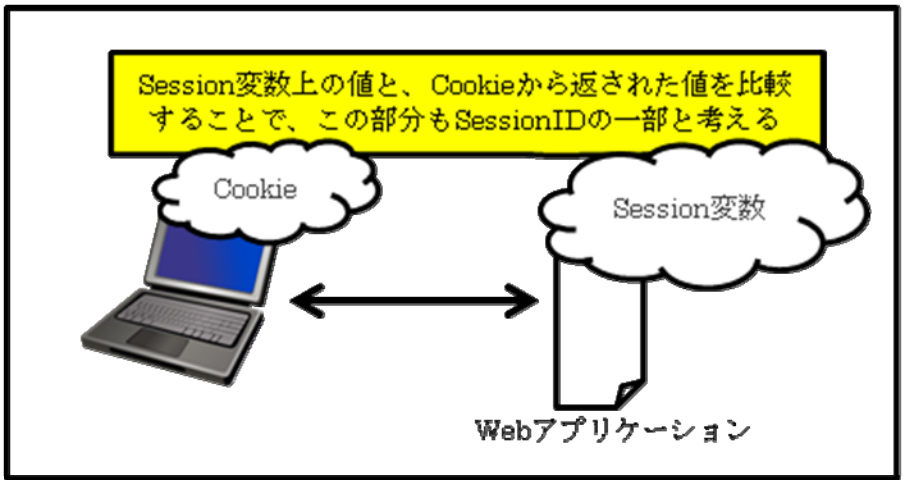


図 6.6-2 : 自作クッキーとセッション変数も含めて SessionID とみなすことで、
フレームワークに実装していないセッション管理の機能を実装させることが可能

Copyright 2010 NTT Communications Corporation

9. 検証作業者

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター
佐名木 智貴
本城 敏信

10. 参考

1. PHP: session_start - Manual
http://www.php.net/manual/ja/function.session-start.php
2. PHP: 実行時設定 - Manual
http://www.php.net/manual/ja/session.configuration.php#ini.session.auto-start
3. PHP: session_regenerate_id - Manual
http://www.php.net/manual/ja/function.session-regenerate-id.php
4. PHP: session_id - Manual
http://www.php.net/manual/ja/function.session-id.php
5. 「CSRF」と「Session Fixation」の諸問題について
http://www.ipa.go.jp/security/vuln/event/documents/20060228_3.pdf
6. Session Fixation Vulnerability in Web-based Applications
http://www.acros.si/papers/session_fixation.pdf
7. Cookiemonster - Cookie Bug Affecting Non-Generic Domains
http://homepages.paradise.net.nz/~glineham/cookiemonster.html
8. Cross Site Cooking
http://www.securityfocus.com/archive/107/423375
9. PHP の Session Adoption は重大な脅威ではない
http://d.hatena.ne.jp/ockeghem/20090515/p1
10. PHP:既知のセキュリティ脆弱性 - Session Adoption
http://blog.ohgaki.net/php-session-adoption
11. session_regeneration_id とセッションフィクシエーション対策
http://www.ecoop.net/memo/2009-08-19-1.html
12. セキュア Web プログラミング Tips 集(出版社:株式会社ソフト・リサーチ・センター)
ISBN=978-4883732562

11. 履歴

- 2010 年 03 月 09 日 : ver1.0 最初の公開

Copyright 2010 NTT Communications Corporation

12. 最新版の公開URL

http://www.icto.jp/security_report/index.html

13. 本レポートに関する問合せ先

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター

e-mail: scan@ntt.com

以 上

Copyright 2010 NTT Communications Corporation