東京デザインテクノロジー専門学校

チーム:GYW

MBSD Cybersecurity Challenges

調査結果報告書

2016年11月

* 1. 調査手法に関する説明

ツールを使った・作った、チームメンバの役割分担、調査手法など、脆弱性発見を網羅的・効率的に実施するために行った工夫があれば、アピールしてください。

※上記欄に収める必要はありません。複数ページで記載いただいて問題ありません。

7

* 1. 発見した脆弱性詳細

1. フレンド以外にメッセージが送信可能

* 対象

メッセージ送信

http://10.128.0.21/sendmail.php

* 危険度

**Medium**

* 解説

メッセージ送信画面では、フレンドのみセレクトボックスで選択出来るようになっているため、フレンドにしかメッセージが送信出来ない仕様と推測される。

しかし、ブラウザの開発者ツールなどを利用し次のようにパラメータ値を書き換えることにより、フレンドではないユーザにメッセージを送信出来ることが確認出来た。

▼再現方法

正常パラメータ：to\_id=3 (フレンドのユーザID）

操作パラメータ：to\_id=10 (フレンドではないユーザのID）

▼条件

フレンドではないユーザのIDを推測する必要がある。

しかし、ユーザIDは連番のため、他ユーザのIDを推測することは容易である。

▼該当箇所

(1)　メッセージ送信

* 想定される被害・影響

フィッシングやspam送信などに悪用される可能性がある。

* 対策

サーバ側のアプリケーションにおいて、指定された送信先(to\_id)がフレンドかどうか確認し、フレンドの場合のみメッセージを送信する。フレンドのIDではない場合はエラーにする。

* 備考

特になし。

1. SNMPデフォルトコミュニティ名の付与

* 対象

検出したホスト（ポート番号）

10.128.0.21(161/udp)

* 危険度

**Low**

* 解説

SNMPコミュニティ名にpublicが設定されている。

publicやprivate等の名称は、SNMPコミュニティ名のデフォルト値としてよく利用されるため、容易に推測される。

* 想定される被害・影響

SNMP MIB情報を取得される。

* 対策

SNMPコミュニティ名を推測しにくい名称に変更する。

* net-snmpの場合

snmpd.confを以下の通り変更する。

例）コミュニティ名をmbsd0123に変更する場合

変更前:

com2sec notConfigUser default public

変更後:

com2sec notConfigUser default mbsd0123

* Microsoft Windows Server(SNMP Service)の場合

1. サービスからSNMP Serviceのプロパティを選択する。
2. セキュリティタブ内、「受け付けるコミュニティ名」の

「削除」、「追加」にて任意のコミュニティ名に変更する。

* その他の場合

対策方法の詳細はベンダに確認する。

* 備考

特になし。

1. 脆弱性名

* 対象

機能名

http://www.xxxx.jp/

* 危険度

**High**

* 解説

脆弱性についての解説を記載

* 想定される被害・影響

想定される被害や影響について記載

* 対策

どのような修正を施せば良いか、対策について記載

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

1. 弱いpasswordを設定できる

* 対象

弱いpassword設定

http://192.168.71.128/register.php

* 危険度

**High**

* 解説

弱いpasswordが推測されやすいから、自分の情報を他人にばれやすい

* 想定される被害・影響

自分のアカウントが他人に不当登録される

* 対策

ユーザーsign up時に、設定されたpasswordをチェックする.

（半角英小文字大文字数字をそれぞれ1種類以上含む8文字以上100文字）のpasswordしか登録できない

**register.phpの98行、変更前：**

if (! isset ( $\_POST ["password"] ) || ! isset ( $\_POST ["password2"] )) {

変更後：

if (! isset ( $\_POST ["password"] ) || ! isset ( $\_POST ["password2"] ) ||

! preg\_match ( ‘/\A(?=.\*?[a-z])(?=.\*?[A-Z])(?=.\*?\d)[a-zA-Z\d]{8,100}+\z/’,

$\_POST ["password "] )) {

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

2.他のユーザーのdiaryをすべて読める

* 対象

他のユーザーのdiaryを読める

http://192.168.71.128/readdiary.php?id=

* 危険度

**High**

* 解説

Readdirry.phpページに、URLのreaddiary.php?id=の所に他ユーザのreaddiaryのidを入力して、他のユーザーのdiaryを読める

* 想定される被害・影響

他のユーザーのdiaryはすべて読まれて、個人情報が漏れる

* 対策

Readdiary.phpページの$\_GET[“id”]をチェックする。$\_GET[“id”]が空の場合は何をやらずにindex.phpに戻る.

Readdiary.phpページの$diaryのwriter\_user\_idをチェックする。writer\_user\_idが自分のidじゃない場合は何をやらずにindex.phpに戻る.

変更前：

$diary = Diary::getDiary ( $\_GET ["id"] );

$writer = $diary->write\_user ();

if ($diary) {

$title = $diary->title == "" ? "(無題)" : $diary->title;

printHeader ();

変更後：

$diary = Diary::getDiary ( $\_GET ["id"] ? $\_GET ["id"] : -1);

If($diary){

If(!$diary->isReadable($user)){

header ( "Location: ./index.php" );

exit ();

}

$title = $diary->title == "" ? "(無題)" : $diary->title;

printHeader ();

}

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

3.他のユーザーのdiaryを編集できる

* 対象

他のユーザーのdiaryを編集できる

Readdiary.phpの編集ボタン

* 危険度

**High**

* 解説

編集ボタンが表示されるとき、デベロッパーツールで編集ボタン関連の<input>のvalueの値を書き換えて、ボタンを押したら、他のユーザーの日記編集画面へ遷移して、他のユーザーの日記を編集できる

* 想定される被害・影響

他のユーザーの情報が漏れて、破れられる

* 対策

$\_POST[“id”]をチェックして、空の場合はindex.phpへ遷移する。存在の場合は$diaryのwrite\_user\_idをチェックして、write\_user\_idが自分の場合は日記を書き込める。

変更前：

else {

$diary = Diary::getDiary ( $\_POST ["id"] );

}

…………………

$diary->save ();

変更後：

else{

$diary = Diary::getDiary ( $\_POST ["id"] ? $\_POST ["id"] : -1);

if (! $diary) {

header ( "location: ./index.php" );

}

if ($diary->isWritable ( $user )) {

…………………

$diary->save ();

}

}

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

4.SQLインジェクションの対策してない

* 対象

SQLインジェクション問題

すべでのSQL命令

* 危険度

**High**

* 解説

想定しないSQL文を実行させて、データベースシステムを不正に操作できる

* 想定される被害・影響

データベースの情報が漏れて、破れられる

* 対策

このシステムの全部のSQL命令を実行する前にphpのバインド機構を利用する。bindValue()関数を利用する

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

5.入力値が特殊文字、意図しない形で表示する

* 対象

入力値を表示するところ

* 危険度

**High**

* 解説

スクリプト挿入攻撃とXSS

GETリクエストのパラメータ中に悪意なscriptを入力して、意図しない結果がページで出る

* 想定される被害・影響

セッションIDの窃取

フォームへの入力内容の窃取

別物のwebページに見せる

* 対策

Htmlspecialcharsで特殊文字をHTMLエンティティに変換する

変更後：

read.phpの28行： <?php echo h($message->message);?>

readdiary.phpの18行：<?php echo h($diary->timestamp); ?>

readdiary.phpの22行：<?php echo h($diary->content); ?>

userlist.phpの144行：

<?php print date("Y/m/d H:i:s", h($diaries[$i]->timestamp));?>

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

6.SESSION\_id更新しない

* 対象

SESSION\_id

* 危険度

**High**

* 解説

セッションハイジャック攻撃を受ける可能性がある

* 想定される被害・影響

セッションIDの窃取

* 対策

ログイン後、session\_regenerate\_id()を利用して、session\_idを更新する

User.phpの63~65行変更前：

if (password\_verify ( $password, $user->password )) {

return $user;

}

変更後：

if (password\_verify ( $password, $user->password )) {

session\_regenerate\_id(true);

return $user;

}

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

7.クラスのフィールドはpublic

* 対象

クラスUser,Message,FriendList,Diaryのフィールド

* 危険度

**low**

* 解説

全部クラスのフィールドはpublicにした

* 想定される被害・影響

フィールドの値をクラス外部から書き換えられる可能性がある

* 対策

クラスUser,Message,FriendList,Diaryのフィールドをprivateにする

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

8.新規登録確認画面passwordが表示される

* 対象

新規登録確認画面のpassword

* 危険度

**High**

* 解説

新規登録確認画面へ遷移して、自分のpasswordが表示される

* 想定される被害・影響

自分のpasswordを他の人に見られる

* 対策

確認画面を表示する時、passwordを隠す

**register.phpの144行の [** <?php print h($\_POST ["password"]) ?> ] を

[ ●●●●●●●● ] で書き換える

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

9.複数な同じなloginid存在できる

* 対象

ユーザーのloginid

* 危険度

**High**

* 解説

同じなloginidを登録できる

* 想定される被害・影響

複数な同じなloginidが存在場合、使えるloginidが一つしかない

useListに複数な同じなユーザーが表示される

* 対策

loginidを新規登録前にloginidが既存するかをチェックする

User.phpのmake()の変更：

make()のparameterに$name ,$genderを二つの変数を加える

43行の$tmp->save();の前に

$tmp->name = $name;

$tmp->gender = $gender;

を加える

Register.phpの63行~70行を以下のコードを書き換える：

$tmp = new User ();

$tmp->make($\_POST[“loginid”] ,password\_hash($\_POST[“password”] ,PASSWORD\_DEFAULT) ,$\_POST[“prof”] ,$\_POST[“name”] ,$\_POST[“gender”]);

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

10.管理ユーザーが一般ユーザーへ連絡できる

* 対象

管理ユーザ：master

* 危険度

**High**

* 解説

管理ユーザmasterが一般ユーザにフレンド申請を出したり、フレンド申請を承認したり、メッセージを送ることできる

* 想定される被害・影響

管理ユーザは一般ユーザ連絡して、詐欺行為可能性がある

* 対策

管理ユーザのフレンド申請、フレンド申請承認、一般ユーザにメッセージを送る機能を取消

上の機能を利用する前に自分が管理ユーザかをチェックする

User.phpのsendFriendRequest(), getFriendRequestUsers(), approvalFrindRequest(), sendMessage()を変更したコード：

public function sendFriendRequest($to\_user) {

if ($this->isFriend ( $to\_user )) {

return false;

}

if(($this->privilege == 0) && ($to\_user->privilege == 0)){

FriendList::makeFriendRequest ( $to\_user, $this );

}

}

public function getFriendRequestUsers() {

if(($this->privilege == 0) && ($to\_user->privilege == 0)){

return FriendList::\_getFriendRequestUsers ( $this );

}

}

public function approvalFrindRequest($form\_user) {

if(($this->privilege == 0) && ($to\_user->privilege == 0)){

FriendList::forceFriend ( $this, $form\_user );

}

}

public function sendMessage($to\_user, $title, $text, $file = "") {

if($this->isFriend($to\_user) && ( $this->privilege == 0 )

&& ($to\_user->privilege == 0)){

$message = new Message ();

$message->to\_user\_id = $to\_user->id;

$message->from\_user\_id = $this->id;

$message->title = $title;

$message->message = $text;

if ($file != "") {

$message->file = $file;

}

$message->save ();

return $message->id;

}

}

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

11.dalatefriend時、stateをチェックしない

* 対象

FriendList.phpのdeleteFriend()

* 危険度

**low**

* 解説

index.php中の拒否ボタンを利用して、開発者ツールでフレンドのidを入力して、フレンドを削除できる

* 想定される被害・影響

他の人に悪意的に自分のフレンドを削除される可能性

* 対策

deleteFriend()を実行する時、stateを判定する変数$isFriendが新しいパラメーターになる

$isFriend = 1 [**$user1と$user2はフレンド**]

$isFriend = 0 [**$user1と$user2はフレンドではない**]

**FriendList.phpの**deleteFriend()、変更前：

**public static function deleteFriend($user1, $user2) {**

**global $db;**

**$statement = $db->query ( "DELETE FROM friends WHERE (to\_user\_id = {$user1->id} AND from\_user\_id = {$user2->id}) OR (to\_user\_id = {$user2->id} AND from\_user\_id = {$user1->id})" );**

**$statement->execute ();**

**}**

**変更後：**

**public static function deleteFriend($user1, $user2,$isFriend) {**

**global $db;**

**if($isFriend == 1){**

**$statement = $db->query ( "DELETE FROM friends WHERE (to\_user\_id = {$user1->id} AND from\_user\_id = {$user2->id} AND state = 1) OR (to\_user\_id = {$user2->id} AND from\_user\_id = {$user1->id} AND state = 1)" );**

**}else{**

**$statement = $db->query ( "DELETE FROM friends WHERE (to\_user\_id = {$user1->id} AND from\_user\_id = {$user2->id} AND state = 0) OR (to\_user\_id = {$user2->id} AND from\_user\_id = {$user1->id} AND state = 0)" );**

**}**

**$statement->execute ();**

**}**

index.phpの15行、変更前：

$user->deleteFriend ( $fromUser );

変更後：

$user->deleteFriend ( $fromUser,0);

User.phpの14行、変更前：

$user->deleteFriend ( User::getUser ( $\_POST ["to\_id"] ) );

変更後：

$user->deleteFriend ( User::getUser ( $\_POST ["to\_id"] ),1);

User.phpのdeleteFriend(),変更前：

public function deleteFriend($target) {

FriendList::deleteFriend ( $this, $target );

}

変更後：

public function deleteFriend($target,$isFriend) {

FriendList::deleteFriend ( $this, $target,$isFriend);

}

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

12.フレンドを検索する時、userListにフレンドが二個が表示される

* 対象

userlist

* 危険度

**Medium**

* 解説

フレンドがいる場合、フレンドを検索して、フレンドが二個表示される。

* 想定される被害・影響

useListがおかしくなる

* 対策

変更前：

if(isset($\_GET["name"])){

//事前にエスケープ

$word = h($\_GET["name"]);

}

$users = User::searchUsers ($word);

$users = array\_merge($users, $user->getFriends());

変更後：

if(isset($\_GET["name"])){

//事前にエスケープ

$word = h($\_GET["name"]);

$users = User::searchUsers ($word);

}else{

$users = $user->getFriends();

}

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

13.アップロードされるファイルのsizeを限定しない

* 対象

アップロードされるファイル

* 危険度

**Medium**

* 解説

sizeを限定しないままファイルをアップロードできる

* 想定される被害・影響

巨大なファイルを連続して送信することによりDoS攻撃

* 対策

ファイルをアップロードする前に、ファイルのsizeを限定する

**sendmessage.phpの19行、変更前：**

if (move\_uploaded\_file($\_FILES['file']['tmp\_name'], "./tmp/".$tname) && preg\_match("/^[^.]+\.jpg$/",$tname)) {

変更後：

if (move\_uploaded\_file($\_FILES['file']['tmp\_name'], "./tmp/".$tname) && preg\_match("/^[^.]+\.jpg$/",$tname && $\_FILES['file'][‘size’] <= “限定したいsize”)) {

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

14.URLからファイル名が丸見えられる

* 対象

URL

* 危険度

**Medium**

* 解説

サーバー中のファイル名がURLに表示される

* 想定される被害・影響

ファイル名がすべてハッカーに見られる

* 対策

URLのファイル名を隠す

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載

15.tokencheck問題

* 対象

formから値を取得する$\_POST

* 危険度

**High**

* 解説

formから$\_POSTで値を取得する時、tokencheck()しない所がある。CSRF攻撃を受ける可能性がある

* 想定される被害・影響

サーバー中の情報が漏れて、破られる

* 対策

送信する際に、tokencheck()でチェックする。

readdiary.phpの34~36行、変更前：

<input type="hidden" name="id" value="<?php echo h($\_GET["id"]); ?>">

<input type="submit" value="編集する" class="btn2">

変更後：

<input type="hidden" name="id" value="<?php echo h($\_GET["id"]); ?>">

<input type="hidden" name="csrf\_token"

value="<?php print$\_SESSION["csrf\_token"]; ?>">

<input type="submit" value="編集する" class="btn2">

changeprofile.phpの11行、変更前：

if ($\_SERVER ["REQUEST\_METHOD"] == "POST") {

変更後：

if ($\_SERVER ["REQUEST\_METHOD"] == "POST" && tokencheck()) {

changeprofile.php に69~70行変更前：

<input type="hidden" name="crsf\_tekon"

value="<?php print sha1(time()); ?>">

変更後：

<input type="hidden" name="crsf\_tekon"

value="<?php echo $\_SESSION["csrf\_token"]; ?>" >

**writediary.phpの10行、変更前：**

if ($\_SERVER ["REQUEST\_METHOD"] == "POST" && isset ( $\_POST ["mode"] )) {

変更後：

if ($\_SERVER ["REQUEST\_METHOD"] == "POST" && isset ( $\_POST ["mode"] )&& tokencheck()) {

**writediary.phpの40行の<form>の中に**

**<input type="hidden" name="csrf\_token" value="<?php echo $\_SESSION["csrf\_token"]; ?>">**

**を入れる**

**index.phpの10行、変更前：**

if (isset ( $\_POST ["from\_id"] )) {

変更後：

if (isset ( $\_POST ["from\_id"] ) && tokencheck()) {

**index.phpの43行の<form>の中に**

**<input type="hidden" name="csrf\_token" value="<?php echo $\_SESSION["csrf\_token"]; ?>">**

**を入れる**

* 備考

その他上記に当てはまらないが報告すべきことがあれば記載