

Bezpieczeństwo — audyt i poprawa aplikacji

Czas: 2 × 45 min

praca indywidualna lub w parach (audyt wspólnie, poprawki indywidualnie)

Cel: Przeprowadzenie przeglądu bezpieczeństwa małej aplikacji PHP według listy kontrolnej i poprawienie kodu tak, aby stosował dobre praktyki obronne.

Po zajęciach potrafisz:

- wyszukuje w kodzie miejsca niezgodne z listą kontrolną i dokumentuje je w tabeli audytu
- zastępuje doklejanie danych do SQL zapytaniami przygotowanymi `mysqli_prepare`
- escapuje dane na wyjściu i wdraża tokeny CSRF w formularzach zmieniających dane
- sprawdza uprawnienia po stronie serwera (właściciel lub admin) i poprawnie kończy skrypt po przekierowaniu
- ukrywa szczegóły błędów przed użytkownikiem i ustawia nagłówki bezpieczeństwa z Content-Security-Policy

Sprzęt i materiały

- komputer z Windows 11, XAMPP, przeglądarka z narzędziami F12
- `r2_14_tablica.sql` i folder `r2_14_start` z Pliki_do_zadan

Oprogramowanie

- **XAMPP**
[apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code** — wyszukiwanie w plikach: Ctrl+Shift+F
code.visualstudio.com
- **OWASP Top 10** — opisy kategorii ryzyk i zalecane zabezpieczenia
owasp.org/Top10

Przygotowanie przed zajęciami

- Przejrzyj wykład 2.11 (zasady obrony) i własny kod z zadania 12.

Bezpieczeństwo

- Pracujesz wyłącznie na własnej kopii aplikacji na `localhost`. Nie testujesz żadnych cudzych stron ani serwerów — to niezgodne z prawem i regulaminem szkoły.
- Celem zadania jest poprawa kodu. Sprawdzasz działanie zwykłymi danymi (np. tytuł z apostrofem lub znakami < >), nie szukasz sposobów obejścia zabezpieczeń.
- Przed zmianami zrób kopię folderu `tablica_przed` — posłuży do porównania.

Przebieg zadania

1 Uruchomienie i audyt

Skopiuj `r2_14_start` do `htdocs\tablica` (i kopię do `htdocs\tablica_przed`), zaimportuj `r2_14_tablica.sql`. Konta: admin / Admin2026!, anna / Anna2026!, piotr / Piotr2026!.

LISTA KONTROLNA

- [1] Dane od użytkownika trafiają do SQL wyłącznie przez zapytania przygotowane.
- [2] Każda dana wypisywana w HTML przechodzi przez `htmlspecialchars` (funkcja `e()`).
- [3] Formularze POST zmieniające dane mają token CSRF; zmiany danych tylko metodą POST.
- [4] Serwer sprawdza zalogowanie i uprawnienia (właściciel/admin); po `header('Location')` jest `exit`.
- [5] Sesja: ciasteczko `HttpOnly` + `SameSite`, `session_regenerate_id(true)` po zalogowaniu.
- [6] Użytkownik nie widzi treści błędów ani zapytań SQL; błędy trafiają do logu.
- [7] Nagłówki: `Content-Security-Policy`, `X-Content-Type-Options`, `Referrer-Policy`, `X-Frame-Options`.

1. Zaloguj się jako anna i przejrzyj aplikację: lista, wyszukiwanie, dodawanie, usuwanie. Zauważ, przy których ogłoszeniach widać „Usuń”.
2. Przeczytaj wszystkie pliki PHP i wypełnij tabelę audytu w sprawozdaniu: plik, nr linii, punkt listy [1]–[7], opis problemu, planowana poprawka. Znajdź co najmniej 12 niezgodności.
3. W F12 → Network obejrzyj nagłówki odpowiedzi `index.php` i zapisz, których z punktu [7] brakuje.

2 Zapytania przygotowane i escapowanie [1] [2]

Utwórz plik `bezpieczenstwo.php` dołączany w `db.php`; w nim funkcja `e()` i pozostałe funkcje z kolejnych kroków.

1. Przepisz wszystkie zapytania z danymi od użytkownika na `mysqli_prepare` + `mysqli_stmt_bind_param` + `mysqli_stmt_get_result` (wyszukiwanie z `LIKE ?` i wzorcem `'%' . $q . '%'`, logowanie, dodawanie, usuwanie). Identyfikator rzutuj/waliduj jako liczbę.
2. Każdą daną z bazy, sesji i formularza wypisuj przez `e()`; treść ogłoszenia dodatkowo `nl2br(e(...))`. Sprawdź: ogłoszenie z tytułem `Test <b>` wyświetla się dosłownie, wyszukiwanie słowa z apostrofem (np. O'Neil) nie powoduje błędu.

3 CSRF, uprawnienia i sesja [3] [4] [5]

```
function csrf_token(): string // losowy token w $_SESSION['csrf'] (random_bytes)
function csrf_pole(): string  // <input type="hidden" name="csrf" value="...">
function sprawdz_csrf(): void  // hash_equals(...), inaczej kod 403 i exit
```

1. Napisz trzy funkcje z ramki i dodaj token do formularzy: logowanie, dodawanie, usuwanie, wylogowanie. W każdej obsłudze POST najpierw wywołaj `sprawdz_csrf()`.
2. Zamień odnośnik „Usuń” na formularz POST z ukrytym `id` i tokenem. `usun.php`: dla GET kod 405, brak ogłoszenia — 404, cudze ogłoszenie — 403 (usuwać może autor albo admin — funkcja `moze_usunac()`); przycisk widzą tylko uprawnieni.
3. Popraw `dodaj.php`: po `header('Location: ...')` dodaj `exit`, waliduj długość tytułu (1–100) i treści (1–2000). Wylogowanie zrób formularzem POST.
4. Ustaw parametry ciasteczka sesji (`HttpOnly`, `SameSite=Lax`) i dodaj `session_regenerate_id(true)` po zalogowaniu. Komunikat błędnego logowania nie powtarza wpisanego loginu.

4 Błędy i nagłówki [6] [7]

1. W `db.php` wyłącz `display_errors`, włącz logowanie do `logs/php_errors.log` (z `logs/.htaccess: Require all denied`), włącz `mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT)` i dodaj `set_exception_handler` z kodem 500 i ogólnym komunikatem. Usuń wszystkie `or die(mysqli_error(...))`. Sprawdź działanie, zmieniając chwilowo nazwę bazy.
2. Na początku `bezpieczenstwo.php` wyślij nagłówki z punktu [7]. CSP: `default-src 'self'; script-src 'self'; style-src 'self'; img-src 'self' data:; object-src 'none'; base-uri 'self'; form-action 'self'; frame-ancestors 'none'`.
3. Z polityką CSP atrybut `onclick` przestanie działać — sprawdź komunikat w konsoli F12. Przenieś potwierdzenie usunięcia do pliku `skrypt.js` (nasłuch `submit` na formularzach `.usun`, dołączony z `defer`).
4. Uzupełnij kolumnę „Poprawiono” w tabeli audytu, wykonaj zrzuty (nagłówki w F12, 403 przy cudzym ogłoszeniu, 405 dla GET, strona błędu 500) i spakuj projekt jako `nazwisko_imie_zad14.zip`.

W sprawozdaniu umieść

- ☐ tabela audytu (co najmniej 12 pozycji) z kolumną „Poprawiono”
- ☐ fragmenty kodu przed i po dla punktów [1], [2], [3] i [4]
- ☐ zrzuty: nagłówki odpowiedzi przed i po, 403 / 405, strona błędu 500, konsola F12 z komunikatem CSP
- ☐ archiwum `nazwisko_imie_zad14.zip`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego zapytanie przygotowane chroni bazę skuteczniej niż ręczne „czyszczenie” danych?
2. Co sprawdza token CSRF i dlaczego musi być losowy i powiązany z sesją?
3. Dlaczego ukrycie przycisku „Usuń” nie wystarcza i serwer musi sprawdzić uprawnienia?
4. Jakie informacje mogłyby zdradzić komunikaty `mysqli_error()` pokazywane użytkownikowi?
5. Do czego służy nagłówek Content-Security-Policy i dlaczego wymaga przeniesienia skryptów inline do plików?

Kryteria oceny

Kryterium	Punkty
Tabela audytu: min. 12 trafnych niezgodności z plikiem, linią i punktem listy	4
[1] Wszystkie zapytania przygotowane; [2] pełne escapowanie wyjścia	4
[3] Token CSRF w 4 formularzach, usuwanie i wylogowanie tylko POST	3
[4] Kontrola uprawnień (403/404/405), <code>exit</code> po przekierowaniu, walidacja długości; [5] sesja	3
[6] Ukryte błędy + log; [7] nagłówki i CSP, skrypt w pliku zewnętrznym	3
Zrzuty, archiwum, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).