

[Dokumentum címe]

[Dokumentum alcíme]



**bevezető – Biztonsági vizsgálat indoklása**

A jelenlegi audit célja a pet shop webalkalmazás biztonsági szempontú átvilágítása. Az alkalmazás forráskódja a GitHub felületén nyilvánosan elérhető (<https://github.com/peterkore/intalk2>), amely önmagában nem sérülékenység, azonban alapvető kiberbiztonsági axióma, hogy:

„A nyilvánosan elérhető forráskód esetén az alkalmazás minden belső működése az ellenfél számára is ismert.”

Ezáltal:

* A „security through obscurity” nem alkalmazható védekezési módszerként.
* A rendszer integritása csak akkor biztosítható, ha az ismert kód ellenére sem áll fenn kihasználható sérülékenység.

Továbbá fontos megemlíteni, hogy az alkalmazás végső célkörnyezete a <https://kivalaszt.hu> domainen lesz elérhető. A weboldalon jelenleg geo-lokációs szűrés van érvényben, amely kizárólag Magyarország, Norvégia és Románia IP-címeiről teszi elérhetővé a tartalmat.

Bár a geo-szűrés egy hasznos kockázatcsökkentő tényező, nem tekinthető elsődleges védelmi vonalnak, mivel:

* VPN vagy proxy használatával megkerülhető,
* nem helyettesíti a belső biztonsági rétegeket (input validáció, jogosultságkezelés stb.).

Ezért minden egyes fájl biztonsági auditálása szükséges. Az alábbi sablon alapján minden fájlt külön oldalon elemzünk.

biztonsági audit fájlonként – Template

Az alábbi sablon szerint készítjük el minden fájl biztonsági áttekintését:

**Fájl neve: példa.php**

**1. Funkció rövid leírása:**  
A fájl célja röviden összefoglalva (pl. beléptető rendszer, termék létrehozás, stb.)

**2. Azonosított sérülékenység:**

Rövid, lényegre törő leírás (pl. SQL injection lehetősége, XSS, jogosulatlan hozzáférés, session kezelés hiánya stb.)

**3. Kockázati szint:**

Alacsony / Közepes / Magas  
(Attól függően, milyen következményei lehetnek a hiba kihasználásának.)

**4. Technikai magyarázat:**

Pontos hivatkozás a problémás kódrészre, és annak működési leírása, valamint miért jelent biztonsági kockázatot.

**5. Javasolt megoldás:**

Konkrét biztonsági javaslat a hiba kijavítására (pl. prepared statement használata, input validáció bevezetése, szerepkör alapú hozzáférés stb.)

**Fájl neve: kosar.php**

**1. Funkció rövid leírása:**  
Egyszerű webáruház kosár megjelenítő oldal, amely listázza a kosár tartalmát, az összesített árat, és lehetőséget ad egyes termékek eltávolítására..

**2. Azonosított sérülékenység:**

**XSS (Cross-site scripting)**

**3. Kockázati szint:**  
**Magas**.

**4. Technikai magyarázat:**

A következő sorban a termék neve közvetlenül kerül kiírásra HTML-ben, anélkül, hogy az kimeneti szűrésen menne keresztül:

|  |
| --- |
| *<?php echo $products[$id]['name']; ?>* |

Ez lehetővé teszi, hogy ha a termék neve tartalmaz ártó szándékú JavaScript kódot (pl. <script>alert('XSS')</script>), akkor az lefusson a felhasználó böngészőjében, amikor a kosarat megtekinti. Ez XSS támadást tesz lehetővé, mellyel például session ellopás, adathalászat vagy felhasználói adatok manipulálása is történhet.

**5. Javasolt megoldás:**

Használjunk **kimeneti szűrést** a HTML-ben való megjelenítés előtt. A PHP htmlspecialchars() függvénye megfelelő védelmet nyújt:

|  |
| --- |
| <?php echo htmlspecialchars($products[$id]['name']); ?> |

Ez a függvény biztonságosan HTML-escape-eli a speciális karaktereket (<, >, ", ', stb.), így a böngésző nem fogja végrehajtani a beágyazott kódot, hanem csak megjeleníti.

**Fájl neve: kosarw.php *(feltételezett név a funkció alapján)***

**1. Funkció rövid leírása:**  
Ez a fájl a webshop kosárfunkcióját kezeli:

* Megjeleníti az adatbázisból lekért termékeket,
* Lehetővé teszi a termékek hozzáadását a kosárhoz,
* Kezeli a kosárból való törlést,
* Megjeleníti az aktuális kosártartalmat és annak összegét.

**2. Azonosított sérülékenység:**

Kliensoldalról érkező adatok megbízhatatlan kezelése (trusting user input)

**3. Kockázati szint:**

Közepes – A felhasználó manipulálhatja az árat vagy termékadatokat, így árhamisítással kárt okozhat a rendszernek.

**4. Technikai magyarázat:**  
A termék hozzáadása a kosárhoz a következőképpen történik:

|  |
| --- |
| $termek\_id = $\_POST['termek\_id'];  $termek\_nev = $\_POST['termek\_nev'];  $termek\_ara = $\_POST['termek\_ara']; |

Ezek az adatok a kliens (böngésző) által küldöttek, tehát teljes mértékben manipulálhatók. Például valaki egyszerűen átírhatja az űrlapban az árat 1 forintra, majd hozzáadja a kosárhoz. Mivel a $\_POST['termek\_ara'] alapján számítódik az összesítés, a rendszer hamis árakat fogad el valós ellenőrzés nélkül.

Ez egy klasszikus ármanipulációs sebezhetőség, amit általában az ún. "hidden field tampering" technikával lehet kihasználni.

**Javasolt megoldás:**

**Szerveroldali árellenőrzés:**

* A termek\_id alapján szerveroldalon *újra* le kell kérdezni a termék valódi adatait az adatbázisból.
* Az árat, készletet és nevet nem a POST kérésből, hanem ebből a lekérdezésből kell beállítani.

Például:

|  |
| --- |
| $id = (int) $\_POST['termek\_id'];  $stmt = $conn->prepare("SELECT termek\_nev, termek\_ara, keszlet FROM webshop WHERE id = ?");  $stmt->bind\_param("i", $id);  $stmt->execute();  $result = $stmt->get\_result();  $termek = $result->fetch\_assoc();  if ($termek) {  $\_SESSION['kosar'][] = [  'id' => $id,  'nev' => $termek['termek\_nev'],  'ara' => $termek['termek\_ara'],  'keszlet' => $termek['keszlet']  ];  } |

**További javaslatok:**

Használj htmlspecialchars() az outputhoz, hogy elkerüld az XSS-t (pl. terméknév megjelenítésénél). Használj CSRF tokeneket az űrlapoknál, mert jelenleg *bárki küldhet POST kérést* a kosár módosítására. Validálj minden bemenetet (pl. az index mezőt is int típusként).

**Fájl neve: megrendw.php**

**1. Funkció rövid leírása:**  
Ez a fájl a megrendelés véglegesítését végzi:

* Frissíti az adatbázisban a termék készletét,
* Kiüríti a kosarat,
* Megjeleníti a kosár tartalmát.

**2. Azonosított sérülékenység:**

SQL injection és logikai hibák a készletfrissítés során

**3. Kockázati szint:**

**Magas** – **A támadó manipulálhatja a lekérdezést és hibás adatokat vihet be (vagy akár az egész adatbázist károsíthatja).**

**4. Technikai magyarázat:**

**Sérülékeny SQL lekérdezés:**

A következő sor:

|  |
| --- |
| $conn->query("UPDATE webshop set keszlet=keszlet-1 where id='$id'"); |

* $id nincs sehol definiálva, így ez egy hibás logika (a változó nem is létezik, hibát dobhat vagy NULL lesz).
* A lekérdezés dinamikusan összefűzött SQL, így ha $id mégis létezne (pl. GET/POST változóként érkezik), akkor SQL injection-re lenne fogékony.

**Ismételt készletfrissítés sablonban (HTML-ben!):**

A HTML foreach ciklusban ez történik:

|  |
| --- |
| $conn->query("UPDATE webshop SET keszlet=keszlet-1 WHERE id=$termek[id]"); |

A PHP sablon renderelés közben módosítja az adatbázist, ami rossz gyakorlat: nem elválasztott felelősségek, nem determinisztikus működés.Ez duplikált készletcsökkentéshez vezethet, hiszen ugyanazt csinálja, amit a checkout POST rész is.

**Input validáció hiánya:**

* A $\_SESSION['kosar'] tartalma nem kerül ellenőrzésre, se típusilag, se logikailag.
* Ha a támadó egy manipulált kosar tömböt helyez el a session-ben (pl. saját POST-tal), akkor bármilyen id-hez le tud vonni készletet.

|  |
| --- |
|  |

**5. Javasolt megoldás:**

Biztonságos készletfrissítés prepared statement-tel:

|  |
| --- |
| $stmt = $conn->prepare("UPDATE webshop SET keszlet = keszlet - 1 WHERE id = ?");  foreach ($\_SESSION['kosar'] as $termek) {  $stmt->bind\_param("i", $termek['id']);  $stmt->execute();  } |

**Ne frissíts adatot HTML-ben!**

Az adatbázis-módosítások csak POST kérés feldolgozásakor történjenek, ne foreach ciklus közben sablonban!

**Tisztázd a logikát:** A checkokut POST rész tartalmazza az összes készletműveletet.A HTML rész csak a megjelenítésért feleljen.

**Input validáció:**

A $\_SESSION['kosar'] minden elemének id mezőjét ellenőrizd, hogy tényleges termék-e.

Pl.:

|  |
| --- |
| $id = (int) $termek['id'];  $result = $conn->query("SELECT COUNT(\*) as count FROM webshop WHERE id = $id");  $row = $result->fetch\_assoc();  if ($row['count'] > 0) {  // mehet az update  } |

**Ne használd print\_r($\_POST) fejlesztési kódrészletet éles környezetben!**

Információt szivárogtat ki, ami segíti a támadót.