

Aktualne zagrożenia cyberbezpieczeństwa wg OWASP

Prof. ALK dr hab. Przemysław Polański
XIV Konferencja Bezpieczeństwo w
Internecie, Hacking, 25 czerwca 2022



Co to jest OWASP?

- Fundacja zajmująca się badaniami nad cyberbezpieczeństwem
- Publikuje listy 10 najważniejszych zagrożeń dla aplikacji internetowych każdego roku
- “Globally recognized by developers as the first step towards more secure coding.”
- <https://owasp.org/www-project-top-ten/>



Ewolucja cyberzagrożeń dla aplikacji internetowych w latach 2017-2021

2017

2021

A01:2017-Injection

A02:2017-Broken Authentication

A03:2017-Sensitive Data Exposure

A04:2017-XML External Entities (XXE)

A05:2017-Broken Access Control

A06:2017-Security Misconfiguration

A07:2017-Cross-Site Scripting (XSS)

A08:2017-Insecure Deserialization

A09:2017-Using Components with Known Vulnerabilities

A10:2017-Insufficient Logging & Monitoring

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

(New) A04:2021-Insecure Design

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

(New) A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring

(New) A10:2021-Server-Side Request Forgery

* From the Survey



1. Uzyskanie dostępu do nieautoryzowanych danych (broken access control)

- Przykładowo, zwykły użytkownik może się zalogować jako administrator i wykonać dowolne działania na stronie www.
- OWASP opisuje 8 sposobów łamania zabezpieczeń i tyleż sposobów uniknięcia tego zagrożenia.
- Pomocne są też fragmenty kodu z komentarzem:

Example Attack Scenarios

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));  
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account.

```
https://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply forces browses to target URLs. Admin rights are required for access to the admin page.

```
https://example.com/app/getappInfo  
https://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin page, this is a flaw.



Common Weakness Enumerations czyli powszechne błędy w odniesieniu do broken access control

1. CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
2. CWE-23 Relative Path Traversal
3. CWE-35 Path Traversal: '.../.../'
4. CWE-59 Improper Link Resolution Before File Access ('Link Following')
5. CWE-200 Exposure of Sensitive Information to an Unauthorized Actor
6. CWE-201 Exposure of Sensitive Information Through Sent Data
7. CWE-219 Storage of File with Sensitive Data Under Web Root
8. CWE-264 Permissions, Privileges, and Access Controls (should no longer be used)
9. CWE-275 Permission Issues
10. CWE-276 Incorrect Default Permissions
11. CWE-284 Improper Access Control
12. CWE-285 Improper Authorization
13. CWE-352 Cross-Site Request Forgery (CSRF)
14. CWE-359 Exposure of Private Personal Information to an Unauthorized Actor
15. CWE-377 Insecure Temporary File
16. CWE-402 Transmission of Private Resources into a New Sphere ('Resource Leak')
17. CWE-425 Direct Request ('Forced Browsing')
18. CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')
19. CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere
20. CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory
21. CWE-540 Inclusion of Sensitive Information in Source Code
22. CWE-548 Exposure of Information Through Directory Listing
23. CWE-552 Files or Directories Accessible to External Parties
24. CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key
25. CWE-601 URL Redirection to Untrusted Site ('Open Redirect')
26. CWE-639 Authorization Bypass Through User-Controlled Key
27. CWE-651 Exposure of WSDL File Containing Sensitive Information
28. CWE-668 Exposure of Resource to Wrong Sphere
29. CWE-706 Use of Incorrectly-Resolved Name or Reference
30. CWE-862 Missing A
31. CWE-863 Incorrect
32. CWE-913 Improper Code Resources
33. CWE-922 Insecure S
34. CWE-1275 Sensitive Attribute



Przykład: CWE-1275: Sensitive Cookie with Improper SameSite Attribute

- **Opis**
- The SameSite attribute for sensitive cookies is not set, or an insecure value is used.
- **Rozszerzony opis**
- The SameSite attribute controls how cookies are sent for cross-domain requests. This attribute may have three values: 'Lax', 'Strict', or 'None'. If the 'None' value is used, a website may create a cross-domain POST HTTP request to another website, and the browser automatically adds cookies to this request. This may lead to Cross-Site-Request-Forgery (CSRF) attacks if there are no additional protections in place (such as CSRF tokens).



Przykład: CWE-1275: Sensitive Cookie with Improper SameSite Attribute

▼ Demonstrative Examples

Example 1

In this example, a cookie is used to store a session ID for a client's interaction with a website. The snippet of code below establishes a new cookie to hold the sessionId.

Example Language: **JavaScript**

(bad code)

```
let sessionId = generateSessionId()
let cookieOptions = { domain: 'example.com' }
response.cookie('sessionId', sessionId, cookieOptions)
```

Since the sameSite attribute is not specified, the cookie will be sent to the website with each request made by the client. An attacker who can potentially perform CSRF attack by using the following malicious page:

Example Language: **HTML**

(attack code)

```
<html>

<form id=evil action="http://local:3002/setEmail" method="POST">
<input type="hidden" name="newEmail" value="abc@example.com" />
</form>
<script>evil.submit()</script>
</html>
```

When the client visits this malicious web page, it submits a '/setEmail' POST HTTP request to the vulnerable website. Since the browser automatically appends the 'sessionId' cookie to the request, the website automatically performs a 'setEmail' action on behalf of the client.

To mitigate the risk, use the sameSite attribute of the 'sessionId' cookie set to 'Strict'.

Example Language: **JavaScript**

(good code)

```
let sessionId = generateSessionId()
let cookieOptions = { domain: 'example.com', sameSite: 'Strict' }
response.cookie('sessionId', sessionId, cookieOptions)
```

▼ Potential Mitigations

Phase: Implementation

Set the SameSite attribute of a sensitive cookie to 'Lax' or 'Strict'. This instructs the browser to apply this cookie only to same-domain requests, which provides a good Defense in Depth. The 'Lax' value is in use, cookies are also sent for top-level cross-domain navigation via HTTP GET, HEAD, OPTIONS, and TRACE methods, but not for other HTTP methods that are more state mutation.

Effectiveness: High

Note: While this mitigation is effective for protecting cookies from a browser's own scripting engine, third-party components or plugins may have their own engines that allow access to be able to use XMLHttpRequest to read the headers directly and obtain the cookie.



2. Błędy związane z kryptografią

- Przykładowo, dane osobowe przekazywane są bez stosowania szyfrowania lub z wykorzystaniem przestarzałych algorytmów
- OWASP opisuje 11 sposobów łamania zabezpieczeń i 15 sposobów uniknięcia tego zagrożenia.
- Pomocne są też fragmenty opisujące typowe ataki

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.

Scenario #2: A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g., at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.

Scenario #3: The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

3. Wstrzyknięcia złośliwego kodu

- Przykładowo, złośliwy kod JavaScript jest uruchamiany w przeglądarce użytkownika, który pozwala na kradzież tożsamości
- *Injection slides down to the third position. 94% of the applications were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3%, and 274k occurrences. Notable Common Weakness Enumerations (CWEs) included are CWE-79: Cross-site Scripting, CWE-89: SQL Injection, and CWE-73: External Control of File Name or Path.*

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getI
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'=1.
For example:

```
http://example.com/app/accountView?id=' or '1'=1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data or even invoke stored procedures.



Ewolucja cyberzagrożeń dla aplikacji internetowych w latach 2017-2021

