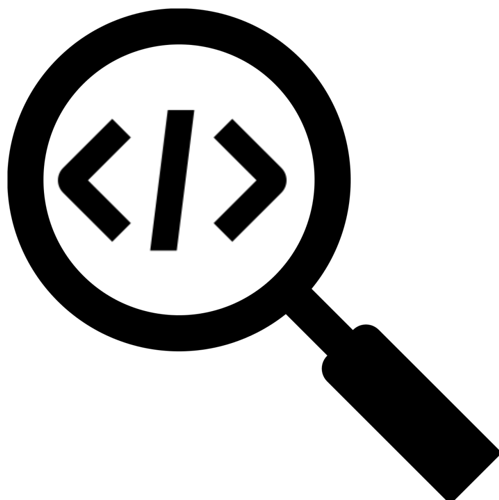
Secure PR reviews: Conviértete en vigilante

SG Virtual Conference 2023

Cristina Mariscal

¿Para qué los “PR reviews”?

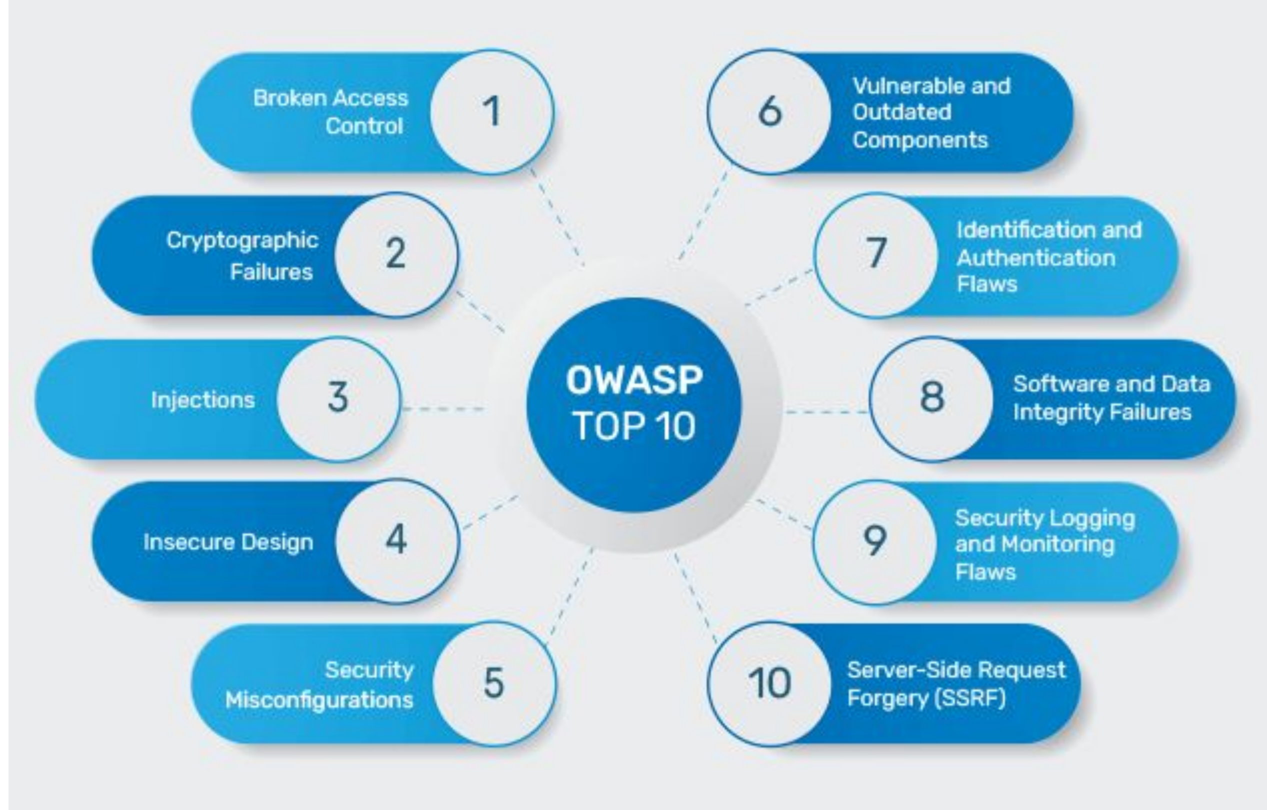


- Las revisiones de código están correlacionados directamente con la reducción de defectos y el hallazgo de vulnerabilidades. [1] [2]
- La probabilidad de encontrar vulnerabilidades en el código incrementa de acuerdo a la cantidad de “reviewers” y su preparación. [2]
- ¿Qué evaluar? Lógica, seguridad, integración, recursos, mejores prácticas.

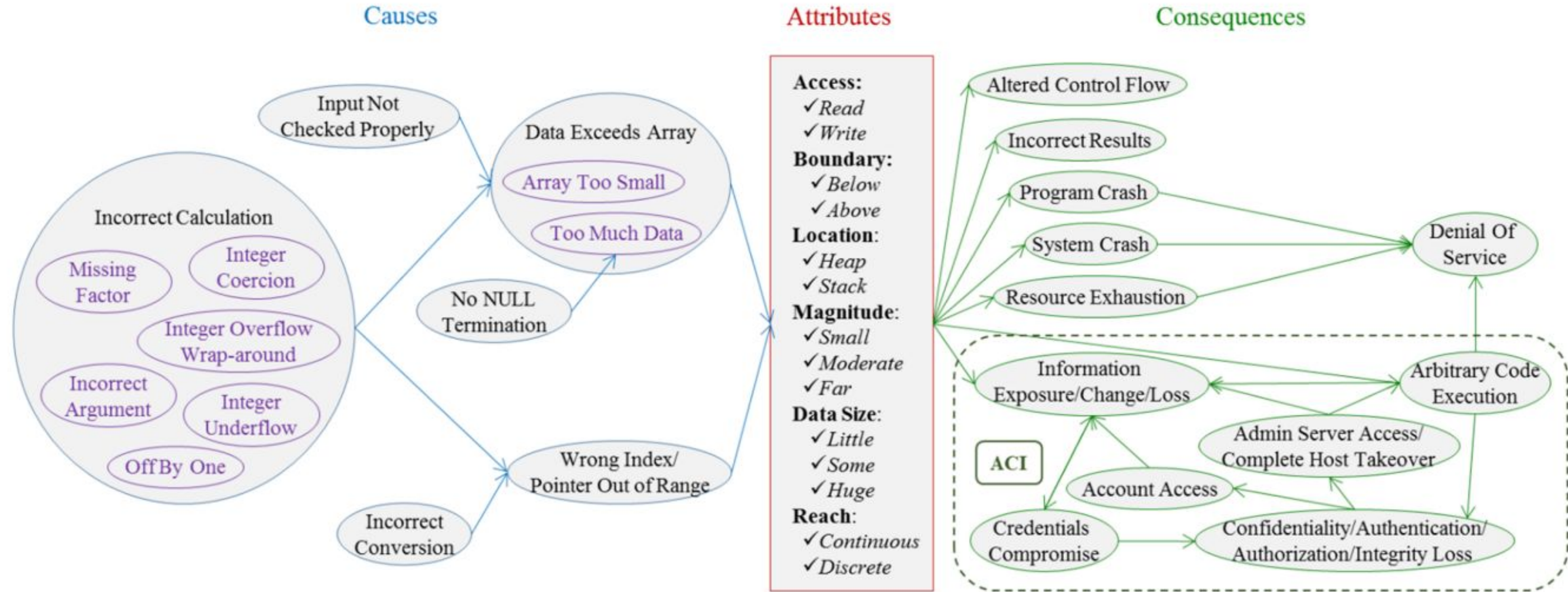
¿Por qué es importante la seguridad en el código?

“ El 64 % de las vulnerabilidades en la base de datos del NIST se deben a errores de programación.

- El 51% de ellos se debieron a errores como Buffer Overflow, Cross-Site Scripting, fallas de inyección, etc. ” [3][4]


OWASP
TOP 10
1
Broken Access Control
2
Cryptographic Failures
3
Injections
4
Insecure Design
5
Security Misconfigurations
6
Vulnerable and Outdated Components
7
Identification and Authentication Flaws
8
Software and Data Integrity Failures
9
Security Logging and Monitoring Flaws
10
Server-Side Request Forgery (SSRF)

Ejemplo de vulnerabilidad: Buffer Overflow



Referencia [6]

1) CVE-2014-0160 – Heartbleed

This vulnerability is listed in [12] and discussed in [14]. Our BF description is:

*Input not checked properly leads to too much data, where a **huge** number of bytes are **read** from the **heap** in a **continuous** reach **after** the array end, which may be exploited for **exposure** of **information** that had not been cleared.*

PR review check list

Lista corta de la Dra M. Greiler: [7]

- ¿A qué vulnerabilidades es susceptible?
- ¿Se manejan la autorización y la autenticación correctamente?
- ¿La entrada (del usuario) es validada correctamente?
- ¿Los datos confidenciales se manejan y almacenan de forma segura?
- ¿Este código NO revela información secreta?
- ¿Los datos obtenidos de fuentes externas son verificados?
- ¿El manejo o registro de errores es apropiado y NO expone al sistema a vulnerabilidades?
- ¿Se utiliza el cifrado correcto?

Áreas de OWASP sobre código seguro: [8]

- Validación de entradas
- Codificación de salidas
- Administración de autenticación y credenciales
- Administración de sesiones
- Control de accesos
- Prácticas de criptografía
- Manejo de errores y logs

- Protección de datos
- Seguridad en las comunicaciones
- Configuración de los sistemas
- Seguridad de bases de datos
- Manejo de archivos
- Manejo de memoria
- Prácticas generales para la codificación

Secure Coding Practices Checklist

Input Validation:

- ☐ Conduct all data validation on a trusted system (e.g., The server)
- ☐ Identify all data sources and classify them into trusted and untrusted. Validate all data from untrusted sources (e.g., Databases, file streams, etc.)
- ☐ There should be a centralized input validation routine for the application
- ☐ Specify proper character sets, such as UTF-8, for all sources of input
- ☐ Encode data to a common character set before validating (*Canonicalize*)
- ☐ All validation failures should result in input rejection
- ☐ Determine if the system supports UTF-8 extended character sets and if so, validate after UTF-8 decoding is completed
- ☐ Validate all client provided data before processing, including all parameters, URLs and HTTP header content (e.g. Cookie names and values). Be sure to include automated post backs from JavaScript, Flash or other embedded code
- ☐ Verify that header values in both requests and responses contain only ASCII characters
- ☐ Validate data from redirects (An attacker may submit malicious content directly to the target of the redirect, thus circumventing application logic and any validation performed before the redirect)
- ☐ Validate for expected data types
- ☐ Validate data range
- ☐ Validate data length

Authentication and Password Management:

- ☐ Require authentication for all pages and resources, except those specifically intended to be public
- ☐ All authentication controls must be enforced on a trusted system (e.g., The server)
- ☐ Establish and utilize standard, tested, authentication services whenever possible
- ☐ Use a centralized implementation for all authentication controls, including libraries that call external authentication services
- ☐ Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control
- ☐ All authentication controls should fail securely
- ☐ All administrative and account management functions must be at least as secure as the primary authentication mechanism
- ☐ If your application manages a credential store, it should ensure that only cryptographically strong one-way salted hashes of passwords are stored and that the table/file that stores the passwords and keys is write-able only by the application. (Do not use the MD5 algorithm if it can be avoided)
- ☐ Password hashing must be implemented on a trusted system (e.g., The server).
- ☐ Validate the authentication data only on completion of all data input, especially for *sequential authentication* implementations
- ☐ Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code
- ☐ Utilize authentication for connections to external systems that involve sensitive information or functions
- ☐ Authentication credentials for accessing services external to the application should be encrypted and stored in a protected location on a trusted system (e.g., The server). The source code is NOT a secure location
- ☐ Use only HTTP POST requests to transmit authentication credentials

Ejemplo práctico 1

login.html ×

login.html > html

```
24    <script>
25      document.getElementById("login").addEventListener("click", LoginHttp)
26      function LoginHttp(){
27        const Http = new XMLHttpRequest();
28        Http.responseType = 'json';
29        let username = document.getElementById("username").value
30        let password = document.getElementById("password").value
31        const url="http://127.0.0.1:8000/login?username="+username+"&password="+password;
32        Http.open("GET", url);
33        Http.send();
34
35        Http.onload = () => {
36          const obj = Http.response;
37          if(obj["succeeded"]){
38            window.location.replace("http://localhost:8081/holi.html?message="+obj["message"]);
39          } else {
40            document.getElementById('message').textContent = obj["message"]
41          }
42        }
43      }
44    </script>
```

Ejemplo práctico 1

```
function LoginHttp(){
  const Http = new XMLHttpRequest();
  Http.responseType = 'json';
  let username = document.getElementById("username").value
  let password = document.getElementById("password").value
  const url="http://127.0.0.1:8000/login?username="+username+"&password="+password;
  Http.open("GET", url);
  Http.send();
```

Ejemplo práctico 2

```
15  @app.get("/login")
16  def read_item(username: str, password: str):
17      return dblogin(username, password)
18
19  def dblogin(username: str, password: str):
20    conn = psycopg2.connect(
21      host="localhost",
22      port=5432,
23      database="TEST060722",
24      user="test",
25      password="123")
26    cur = conn.cursor()
27    cur.execute(f"select 1 from users where username='{username}'")
28    res = cur.fetchone()
29    if not res:
30          return { "succeeded": False, "message": f"User {username} is not registered" }
31    cur = conn.cursor()
32    cur.execute(f"select 1 from users where username='{username}' and password='{password}'")
33    res = cur.fetchone()
34    if not res:
35          return { "succeeded": False, "message": f"Invalid password for {username}" }
36    return { "succeeded": True, "message": f"Valid login {username}" }
```

Ejemplo práctico 2

```python
def dblogin(username: str, password: str):
    conn = psycopg2.connect(
        host="localhost",
        port=5432,
        database="TEST060722",
        user="test",
        password="123")
    cur = conn.cursor()
```

Ejemplo práctico 2

```
cur.execute(f"select 1 from users where username='{username}'")
res = cur.fetchone()
if not res:
    return { "succeeded": False, "message": f"User {username} is not registered" }
```

Ejemplo práctico 2

requirements.txt ×

REST > requirements.txt

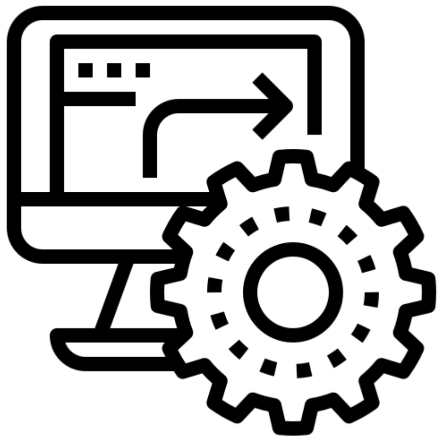
```
1  fastapi==0.78.0
2  uvicorn==0.18.2
3  psycopg2==1.3.8
```

fastapi 0.95.0

uvicorn 0.21.1

psycopg2 2.9.5

Herramientas de análisis de código



“El análisis de código estático buscan posibles vulnerabilidades dentro del código fuente estático (no en ejecución) mediante el uso de técnicas como el análisis de corrupción y el análisis de flujo de datos”. [5]

- Herramientas: ReSharper, SonarQube, CodeScan, DevSkim…

Recursos útiles

- OWASP Secure Coding Practices: Link
- OWASP top ten: Link
- OWASP Secure Code Dojo: Link
- SANS top 25: Link
- CWE top 25: Link
- SEI CERT Coding standards: Link

Referencias

[1] M. Greiler. (2021). “Secure code review handout”. MichaelGreiler. Link.
[2] A. Edmundson et al. (2013). “An Empirical Study on the Effectiveness of Security Code Review”. Springer. Link.
[3] R. Schiela. (2022). “Secure Coding overview”. Carnegie Mellon University. Link.
[4] J. Heffley, P. Meunier. (2004) “Can source code auditing software identify common vulnerabilities and be used to evaluate software security?”. IEEE. Link.
[5] R. Dewhurst. (2023). “Static Code Analysis”. OWASP. Link.
[6] I. Bojanova et al. (2016). “The Bugs Framework (BF): A Structured Approach to Express Bugs”. IEEE. Link.
[7] M. Greiler. (2023). “Secire code review checklist”. AwesomeCodeReviews. Link.
[8] OWASP. (2010). “OWASP Secure Coding Practices. Quick Reference Guide”. Link.