UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Security by Design

Concepts Fondamentaux en Cybersécurité

Confidentialité, Intégrité, Disponibilité

Objectifs

- Comprendre les principes fondamentaux du **Security by Design** et leur importance dans la conception des systèmes
- Identifier les **vulnérabilités** dès la phase de conception et proposer des **contre-mesures** efficaces
- Intégrer les **bonnes pratiques de sécurité** dans le cycle de vie du développement logiciel (SDLC)
- Appliquer les principes de **défense en profondeur**, **séparation des privilèges**, et **least privilege**
- Respecter les **cadres réglementaires** (RGPD, ISO 27001, etc.)

Programme de la Formation

1 - Fondamentaux

- La triade CIA
- Principes du Security by Design
- Menaces et vulnérabilités

2 - Conception Sécurisée

- Defense in Depth
- Least Privilege
- Séparation des privilèges

3 - SDLC Sécurisé

- Intégration dans le cycle de vie
- Threat Modeling
- Secure Coding en Java

4 - Cadres Réglementaires

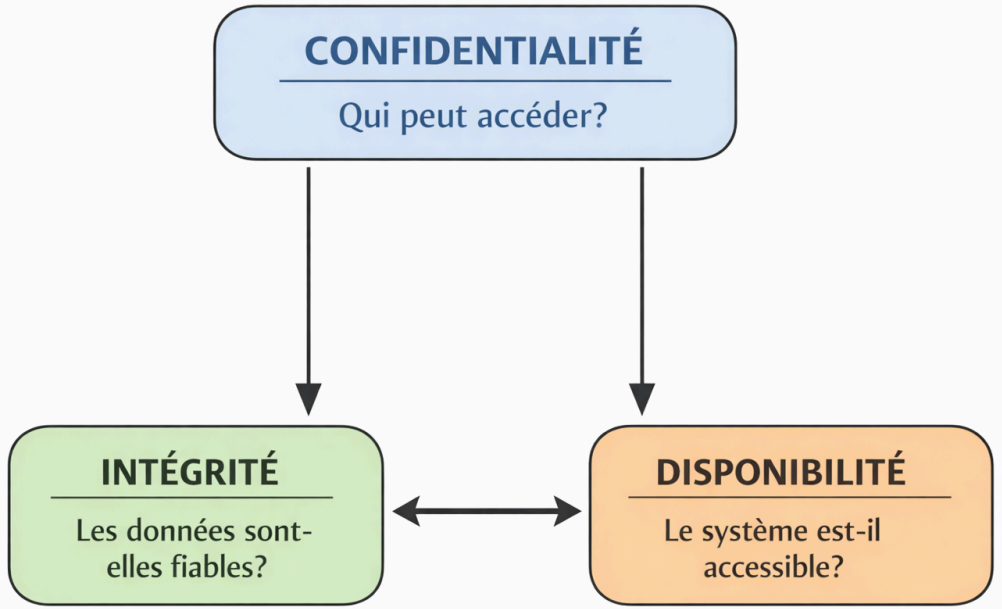
- RGPD et protection des données
- ISO 27001
- OWASP Top 10

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

La Triade CIA

Confidentialité - Intégrité - Disponibilité

La Triade CIA : Vue d'Ensemble



La **triade CIA** est le fondement de toute politique de sécurité informatique

Confidentialité (Confidentiality)

Définition

Garantir que l'information n'est accessible qu'aux personnes autorisées

Menaces

- Écoute réseau (sniffing)
- Vol de données
- Ingénierie sociale
- Accès non autorisé

Contre-mesures

- Chiffrement (AES, RSA)
- Contrôle d'accès (RBAC)
- Authentification forte
- Classification des données

Démo Java : Chiffrement AES

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

public class ConfidentialiteDemo {
    public static void main(String[] args) throws Exception {
        // Génération d'une clé AES 256 bits
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(256);
        SecretKey secretKey = keyGen.generateKey();

        String message = "Données confidentielles à protéger";

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

        System.out.println("Message chiffré: " +
            Base64.getEncoder().encodeToString(encrypted));
    }
}
```

Intégrité (Integrity)

Définition

Garantir que les données n'ont pas été altérées de manière non autorisée

Menaces

- Modification non autorisée
- Injection SQL/XSS
- Man-in-the-Middle
- Corruption de données

Contre-mesures

- Signatures numériques
- Fonctions de hachage
- Contrôles de version
- Validation des entrées

Démo Java : Vérification d'Intégrité avec SHA-256

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

public class IntegriteDemo {
    public static String calculerHash(String data) throws Exception {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(data.getBytes("UTF-8"));
        return Base64.getEncoder().encodeToString(hash);
    }

    public static void main(String[] args) throws Exception {
        String document = "Contrat important - Version 1.0";
        String hashOriginal = calculerHash(document);

        System.out.println("Hash original: " + hashOriginal);

        // Simulation d'une modification
        String documentModifie = "Contrat important - Version 1.1";
        String hashModifie = calculerHash(documentModifie);

        System.out.println("Hash modifié: " + hashModifie);
        System.out.println("Intégrité compromise: " +
            !hashOriginal.equals(hashModifie));
    }
}
```

Disponibilité (Availability)

Définition

Garantir que les systèmes et données sont accessibles quand nécessaire

Menaces

- Attaques DDoS
- Pannes matérielles
- Catastrophes naturelles
- Erreurs de configuration

Contre-mesures

- Redondance (clusters)
- Plans de reprise (PRA/PCA)
- Load balancing
- Sauvegardes régulières

Démo Java : Pattern Circuit Breaker

```java
public class CircuitBreaker {
    private int failureCount = 0;
    private final int threshold = 3;
    private boolean isOpen = false;
    private long lastFailureTime = 0;
    private final long resetTimeout = 30000; // 30 secondes

    public <T> T execute(ServiceCall<T> call) throws Exception {
        if (isOpen && System.currentTimeMillis() - lastFailureTime < resetTimeout) {
            throw new ServiceUnavailableException("Circuit ouvert");
        }

        try {
            T result = call.execute();
            reset();
            return result;
        } catch (Exception e) {
            recordFailure();
            throw e;
        }
    }

    private void recordFailure() {
        failureCount++;
        lastFailureTime = System.currentTimeMillis();
        if (failureCount >= threshold) isOpen = true;
    }
}
```

Équilibre de la Triade CIA

Priorité	Exemple de Système	Justification
C > I > D	Système bancaire	Secrets financiers critiques
I > C > D	Système médical	Exactitude des diagnostics
D > C > I	Service d'urgence	Accessibilité 24/7 vitale
Équilibré	E-commerce	Tous les aspects importants

L'équilibre dépend du **contexte métier** et de l'**analyse des risques**

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

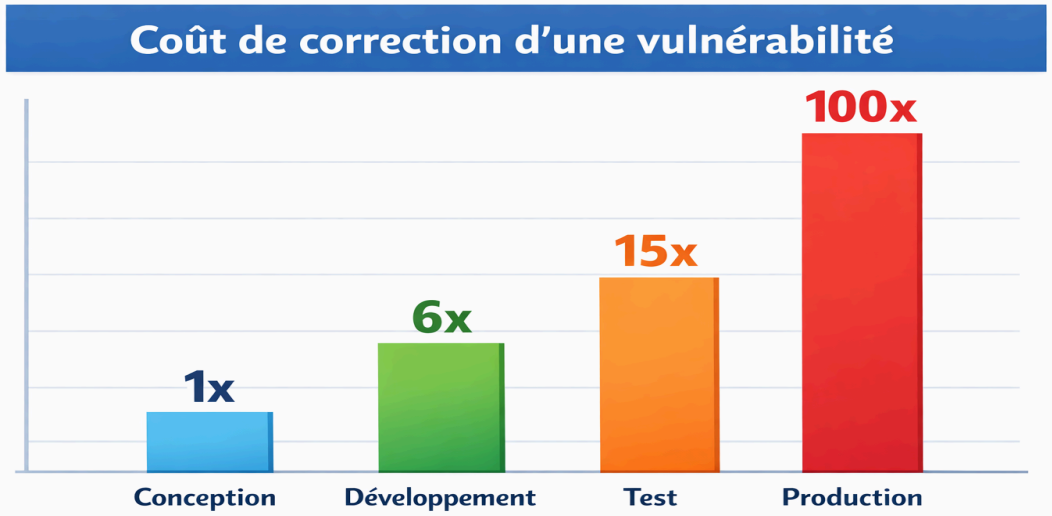
Principes du Security by Design

Qu'est-ce que le Security by Design ?

Définition :

Security by Design : Intégrer la sécurité dès les premières phases de conception d'un système, plutôt que de l'ajouter après coup.

Pourquoi ?



Les 10 Principes du Security by Design

1. **Minimiser la surface d'attaque**
2. **Secure Defaults**
3. **Principe du moindre privilège**
4. **Défense en profondeur**
5. **Fail Securely**
6. **Ne pas faire confiance aux services**
7. **Séparation des responsabilités**
8. **Éviter la sécurité par l'obscurité**
9. **Simplicité**
10. **Corriger correctement les vulnérabilités**

Principe 1 : Minimiser la Surface d'Attaque

Concept

Réduire le nombre de points d'entrée potentiels pour un attaquant.

Mauvaise pratique :

```
// Exposer tous les endpoints par défaut
@RestController
public class AdminController {
    @GetMapping("/api/admin/users")      // Exposé
    @GetMapping("/api/admin/config")     // Exposé
    @GetMapping("/api/admin/logs")       // Exposé
    @GetMapping("/api/admin/database")   // Exposé - DANGEREUX!
}
```

Principe 1 : Minimiser la Surface d'Attaque

Bonne pratique :

```
@RestController
@RequestMapping("/api/admin")
@PreAuthorize("hasRole('ADMIN')")
public class AdminController {

    // Seuls les endpoints nécessaires sont exposés
    @GetMapping("/users")
    public List<UserDTO> getUsers() {
        return userService.getAllUsers();
    }

    // Endpoints sensibles désactivés en production
    @Profile("!production")
    @GetMapping("/debug")
    public DebugInfo getDebugInfo() { ... }
}
```

Principe 2 : Secure Defaults (Valeurs par Défaut Sécurisées)

Concept

Les configurations par défaut doivent être les plus sécurisées possible.

```
public class SecurityConfig {
    // MAUVAIS : Tout permis par défaut
    private boolean requireAuthentication = false;
    private boolean enableEncryption = false;
    private int sessionTimeout = Integer.MAX_VALUE;

    // BON : Tout sécurisé par défaut
    private boolean requireAuthentication = true;
    private boolean enableEncryption = true;
    private int sessionTimeout = 1800; // 30 minutes
    private boolean httpsOnly = true;
    private String csrfProtection = "enabled";
}
```

Principe 3 : Least Privilege (Moindre Privilège)

Concept

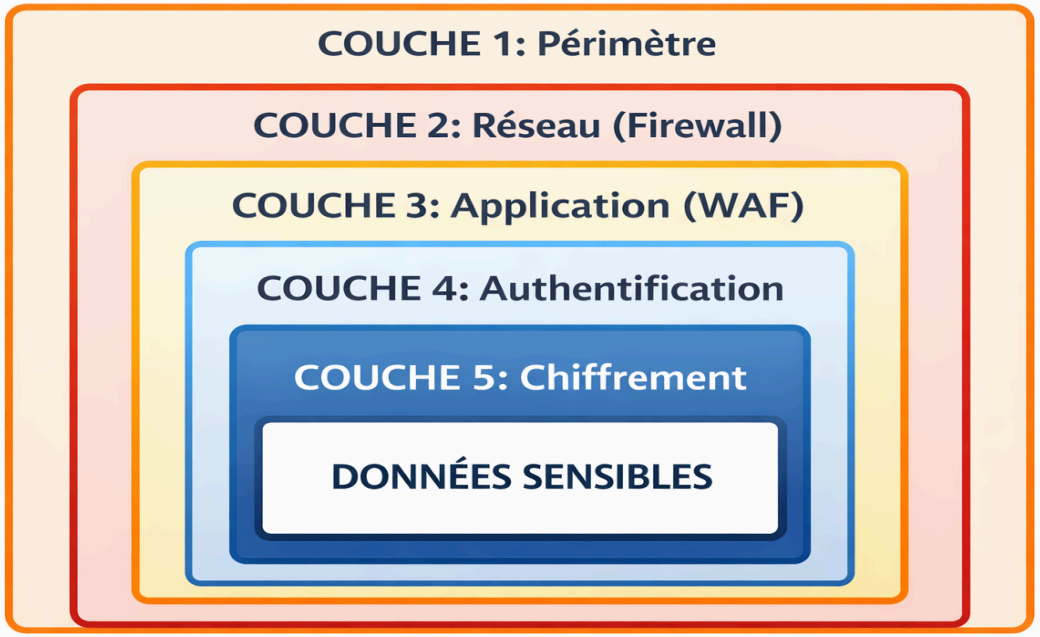
Un utilisateur/processus ne doit avoir que les droits **strictement nécessaires** à sa tâche.

```
// MAUVAIS : Connexion avec compte administrateur
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost/db", "root", "password");

// BON : Compte avec privilèges limités
public class DatabaseConfig {
    // Compte lecture seule pour les rapports
    @Bean("readOnlyDataSource")
    public DataSource readOnlyDataSource() {
        return DataSourceBuilder.create()
            .username("app_readonly")
            .password(vault.getSecret("db_readonly_pwd"))
            .build();
    }

    // Compte avec droits limités pour l'application
    @Bean("appDataSource")
    public DataSource appDataSource() {
        return DataSourceBuilder.create()
            .username("app_user")  // Pas de DROP, pas de GRANT
            .build();
    }
}
```

Principe 4 : Defense in Depth (Défense en Profondeur)

Concept

Plusieurs couches de sécurité indépendantes pour protéger les actifs.



Démo Java : Defense in Depth

```java
@RestController
public class SecureController {

    @PostMapping("/api/transfer")
    @PreAuthorize("hasRole('USER')")           // Couche 1: Autorisation
    @RateLimited(requests = 10, period = 60)   // Couche 2: Rate limiting
    public ResponseEntity<TransferResult> transfer(
            @Valid @RequestBody TransferRequest request,  // Couche 3: Validation
            @AuthenticationPrincipal User user) {

        // Couche 4: Vérification métier
        if (!accountService.belongsToUser(request.getFromAccount(), user)) {
            throw new UnauthorizedException("Compte non autorisé");
        }

        // Couche 5: Audit et logging
        auditService.log(user, "TRANSFER", request);

        // Couche 6: Transaction sécurisée
        return transactionService.executeSecurely(request);
    }
}
```

Principe 5 : Fail Securely (Échec Sécurisé)

Concept

En cas d'erreur, le système doit basculer vers un état sécurisé.

Échec non sécurisé

```
public boolean isAuthorized(User u) {
    try {
        return authService.check(u);
    } catch (Exception e) {
        // En cas d'erreur, on autorise
        return true; // DANGEREUX!
    }
}
```

Échec sécurisé

```
public boolean isAuthorized(User u) {
    try {
        return authService.check(u);
    } catch (Exception e) {
        logger.error("Auth error", e);
        // En cas d'erreur, on refuse
        return false; // SÉCURISÉ
    }
}
```

Principe 6 : Ne Pas Faire Confiance aux Services

Concept

Valider toutes les données provenant de services externes.

```
public class ExternalServiceClient {

    public UserData fetchUserFromExternalAPI(String userId) {
        ResponseEntity<UserData> response = restTemplate
            .getForEntity(externalApiUrl + "/users/" + userId, UserData.class);

        UserData userData = response.getBody();

        // Valider les données reçues même d'un service "de confiance"
        validateUserData(userData);

        // Sanitizer les données
        userData.setName(sanitize(userData.getName()));
        userData.setEmail(validateEmail(userData.getEmail()));

        return userData;
    }

    private void validateUserData(UserData data) {
        Objects.requireNonNull(data, "UserData ne peut pas être null");
        if (data.getId() == null || data.getId().isEmpty()) {
            throw new ValidationException("ID utilisateur invalide");
        }
    }
}
```

Principe 7 : Séparation des Responsabilités

Concept

Diviser les fonctionnalités pour limiter l'impact d'une compromission.

```
// Architecture avec séparation des responsabilités
public class SecureArchitecture {

    // Service 1: Authentification uniquement
    private final AuthenticationService authService;

    // Service 2: Autorisation uniquement
    private final AuthorizationService authzService;

    // Service 3: Audit uniquement
    private final AuditService auditService;

    // Service 4: Opérations métier
    private final BusinessService businessService;

    public Result performAction(Request request) {
        User user = authService.authenticate(request.getCredentials());
        authzService.checkPermission(user, request.getAction());
        auditService.logAction(user, request);
        return businessService.execute(request);
    }
}
```

Principe 8 : Éviter la Sécurité par l'Obscurité

Concept

La sécurité ne doit pas reposer sur le secret de l'implémentation.

Sécurité par l'obscurité

```
// "Personne ne devinera cette URL"
@GetMapping("/x7k9m2p4/admin")
public AdminPanel getAdmin() { }

// "Mon algorithme secret"
public String encrypt(String s) {
    return reverse(rot13(s));
}
```

Sécurité réelle

```
// URL claire + authentification
@GetMapping("/admin")
@PreAuthorize("hasRole('ADMIN')")
public AdminPanel getAdmin() { }

// Algorithme standard + clé secrète
public String encrypt(String s) {
    return AES.encrypt(s, secretKey);
}
```

Principe 9 : Simplicité (KISS)

Concept

Un code simple est plus facile à auditer et contient moins de bugs.

```
// Complexe et difficile à auditer
public boolean checkAccess(User u, Resource r, Action a, Context c) {
    return (u.getRoles().stream().anyMatch(role ->
        role.getPermissions().stream().anyMatch(p ->
            p.getResources().contains(r.getType()) &&
            p.getActions().contains(a) &&
            (!p.hasConditions() || p.getConditions().stream()
                .allMatch(cond -> cond.evaluate(c))))));
}

// Simple et lisible
public boolean checkAccess(User user, Resource resource, Action action) {
    Permission required = new Permission(resource, action);
    return user.hasPermission(required);
}
```

Principe 10 : Corriger Correctement les Vulnérabilités

Concept

Traiter la cause racine, pas juste les symptômes.

```
// Correction superficielle
// Bug: SQL Injection sur le paramètre "name"
public User findUser(String name) {
    // "Correction": filtrer quelques caractères
    name = name.replace("'", "").replace(";", "");
    return jdbc.query("SELECT * FROM users WHERE name='" + name + "'");
}

// Correction à la racine
public User findUser(String name) {
    // Utilisation de requêtes préparées
    return jdbcTemplate.queryForObject(
        "SELECT * FROM users WHERE name = ?",
        new Object[]{name},
        userRowMapper
    );
}
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

SDLC Sécurisé

Intégration de la Sécurité dans le Cycle de Vie

Secure SDLC : Vue d'Ensemble

SECURE SDLC					
EXIGENCES	CONCEPTION	CODE	TEST	DEPLOY	MAINTIEN
Security Require-ments	Threat Modeling	Secure Coding Review	SAST/DAST Pentest	Security Config Hardening	Patch Management Monitoring

Phase 1 : Exigences de Sécurité

Activités Clés

Activité	Description	Livrable
Identification des actifs	Données sensibles, fonctions critiques	Inventaire des actifs
Classification des données	Niveau de sensibilité	Matrice de classification
Exigences de sécurité	Contraintes techniques	Document d'exigences
Analyse des risques	Menaces potentielles	Registre des risques

Exemple : Matrice de Classification des Données

```
public enum DataClassification {
    PUBLIC("Données publiques", SecurityLevel.LOW),
    INTERNAL("Données internes", SecurityLevel.MEDIUM),
    CONFIDENTIAL("Données confidentielles", SecurityLevel.HIGH),
    RESTRICTED("Données restreintes", SecurityLevel.CRITICAL);

    private final String description;
    private final SecurityLevel level;

    // Application dans le code
    public static DataClassification classify(Object data) {
        if (data instanceof PersonalData) return CONFIDENTIAL;
        if (data instanceof FinancialData) return RESTRICTED;
        if (data instanceof InternalDocument) return INTERNAL;
        return PUBLIC;
    }
}
```

Phase 2 : Threat Modeling (Modélisation des Menaces)

Méthodologie STRIDE

Menace	Description	Propriété CIA
Spoofing	Usurpation d'identité	Authentification
Tampering	Modification des données	Intégrité
Repudiation	Nier une action	Non-répudiation
Information Disclosure	Fuite d'informations	Confidentialité
Denial of Service	Indisponibilité	Disponibilité
Elevation of Privilege	Élévation de droits	Autorisation

Démo Java : Annotation pour Threat Modeling

```java
// Annotations personnalisées pour documenter les menaces
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ThreatModel {
    Threat[] threats();
    String[] mitigations();
}

@RestController
public class PaymentController {

    @ThreatModel(
        threats = {Threat.SPOOFING, Threat.TAMPERING, Threat.REPUDIATION},
        mitigations = {
            "MFA obligatoire",
            "Signature des transactions",
            "Audit logging complet"
        }
    )
    @PostMapping("/payment")
    public PaymentResult processPayment(@Valid PaymentRequest request) {
        // Implémentation avec les mitigations
    }
}
```

Phase 3 : Secure Coding

Règles Fondamentales

Validation des Entrées

- Valider toutes les entrées
- Liste blanche > Liste noire
- Encoder les sorties

Gestion des Erreurs

- Ne pas exposer les détails
- Logger de manière sécurisée
- Fail secure

Authentification

- Mots de passe hashés (bcrypt)
- Sessions sécurisées
- Protection contre brute-force

Cryptographie

- Algorithmes standards
- Gestion sécurisée des clés
- TLS 1.3 minimum

Démo Java : Validation des Entrées

```java
@RestController
public class UserController {

    @PostMapping("/register")
    public ResponseEntity<User> register(
            @Valid @RequestBody UserRegistrationDTO dto) {

        // Validation supplémentaire métier
        validateBusinessRules(dto);

        return ResponseEntity.ok(userService.register(dto));
    }
}

public class UserRegistrationDTO {
    @NotBlank(message = "Le nom est obligatoire")
    @Size(min = 2, max = 50)
    @Pattern(regexp = "^[a-zA-ZÀ-ÿ\\s-]+$",
             message = "Caractères non autorisés")
    private String name;

    @Email(message = "Email invalide")
    @NotBlank
    private String email;

    @StrongPassword  // Annotation personnalisée
    private String password;
}
```

Démo Java : Hashage Sécurisé des Mots de Passe

```java
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Service
public class PasswordService {

    // BCrypt avec coût de 12 (recommandé)
    private final BCryptPasswordEncoder encoder =
        new BCryptPasswordEncoder(12);

    public String hashPassword(String plainPassword) {
        // Validation du mot de passe
        validatePasswordStrength(plainPassword);

        // Hashage avec salt automatique
        return encoder.encode(plainPassword);
    }

    public boolean verifyPassword(String plainPassword, String hashedPassword) {
        return encoder.matches(plainPassword, hashedPassword);
    }

    private void validatePasswordStrength(String password) {
        if (password.length() < 12) {
            throw new WeakPasswordException("Minimum 12 caractères");
        }
        // Vérifications supplémentaires...
    }
}
```

Phase 4 : Tests de Sécurité

Types de Tests

TESTS DE SÉCURITÉ

SAST (Static AST)	DAST (Dynamic AST)	PENTEST (Test d'intrusion)
Analyse du code source	Test application en exécution	Simulation d'attaque réelle
Outils: ✓ SonarQube ✓ SpotBugs ✓ Checkmarx	**Outils:** ✓ OWASP ZAP ✓ Burp Suite ✓ Nikto	**Méthodologies:** ✓ OWASP Testing Guide ✓ PTES ✓ OSSTMM

Démo Java : Tests de Sécurité Automatisés

```java
@SpringBootTest
@AutoConfigureMockMvc
public class SecurityTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testSQLInjectionPrevention() throws Exception {
        String maliciousInput = "'; DROP TABLE users; --";

        mockMvc.perform(get("/api/users/search")
                .param("name", maliciousInput))
            .andExpect(status().isBadRequest()); // Doit être rejeté
    }

    @Test
    public void testXSSPrevention() throws Exception {
        String xssPayload = "<script>alert('XSS')</script>";

        MvcResult result = mockMvc.perform(post("/api/comments")
                .content("{\"text\":\"" + xssPayload + "\"}")
                .contentType(MediaType.APPLICATION_JSON))
            .andReturn();

        // Vérifier que le contenu est encodé
        assertFalse(result.getResponse().getContentAsString()
            .contains("<script>"));
    }
}
```

Phase 5 : Déploiement Sécurisé

Checklist de Sécurité

Vérification	Action
Configuration serveur	Désactiver services inutiles
Headers HTTP	CSP, HSTS, X-Frame-Options
Secrets	Utiliser un vault (HashiCorp, AWS SM)
HTTPS	TLS 1.3, certificats valides
Monitoring	Logs de sécurité, alertes
Backup	Sauvegardes chiffrées

Démo Java : Configuration Spring Security

```java
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            // Headers de sécurité
            .headers(headers -> headers
                .contentSecurityPolicy(csp ->
                    csp.policyDirectives("default-src 'self'"))
                .frameOptions(frame -> frame.deny())
                .httpStrictTransportSecurity(hsts ->
                    hsts.maxAgeInSeconds(31536000).includeSubDomains(true))
            )
            // CSRF protection
            .csrf(csrf -> csrf.csrfTokenRepository(
                CookieCsrfTokenRepository.withHttpOnlyFalse()))
            // Autorisation
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/public/**").permitAll()
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            );
        return http.build();
    }
}
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Vulnérabilités Courantes et Contre-mesures

Focus OWASP Top 10

OWASP Top 10 (2021)

#	Vulnérabilité	Description
A01	Broken Access Control	Contrôle d'accès défaillant
A02	Cryptographic Failures	Mauvaise cryptographie
A03	Injection	SQL, NoSQL, OS, LDAP injection
A04	Insecure Design	Conception non sécurisée
A05	Security Misconfiguration	Mauvaise configuration
A06	Vulnerable Components	Composants vulnérables
A07	Auth Failures	Authentification défaillante
A08	Data Integrity Failures	Intégrité des données
A09	Logging Failures	Journalisation insuffisante
A10	SSRF	Server-Side Request Forgery

A01 : Broken Access Control

Vulnérabilité

Accès non autorisé à des ressources ou fonctions.

```
// VULNÉRABLE : IDOR (Insecure Direct Object Reference)
@GetMapping("/api/users/{userId}/documents")
public List<Document> getDocuments(@PathVariable Long userId) {
    // Pas de vérification que l'utilisateur courant
    // a le droit d'accéder aux documents de userId
    return documentService.findByUserId(userId);
}
```

A01 : Correction

```
// SÉCURISÉ : Vérification de l'autorisation
@GetMapping("/api/users/{userId}/documents")
@PreAuthorize("@securityService.canAccessUserDocuments(#userId)")
public List<Document> getDocuments(
        @PathVariable Long userId,
        @AuthenticationPrincipal UserDetails currentUser) {

    // Double vérification
    if (!userId.equals(currentUser.getId())
            && !currentUser.hasRole("ADMIN")) {
        throw new AccessDeniedException("Accès non autorisé");
    }

    return documentService.findByUserId(userId);
}
```

A03 : Injection

Types d'Injections

TYPES D'INJECTION				
SQL SELECT * FROM ...	NoSQL {$gt: ""}	LDAP *)(&(...))	OS ; rm -rf /	XSS <script>

A03 : Prévention des Injections SQL

Vulnérable

```
String query = "SELECT * FROM users " +
    "WHERE name = '" + name + "'";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

Sécurisé

```
String query = "SELECT * FROM users " +
    "WHERE name = ?";
PreparedStatement pstmt =
    conn.prepareStatement(query);
pstmt.setString(1, name);
ResultSet rs = pstmt.executeQuery();
```

A03 : Prévention XSS

```
@Component
public class XSSFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException, ServletException {
        chain.doFilter(new XSSRequestWrapper((HttpServletRequest) request),
                       response);
    }
}

public class XSSRequestWrapper extends HttpServletRequestWrapper {

    @Override
    public String getParameter(String name) {
        String value = super.getParameter(name);
        return sanitize(value);
    }

    private String sanitize(String value) {
        if (value == null) return null;
        // Utiliser OWASP Java Encoder
        return Encode.forHtml(value);
    }
}
```

A02 : Cryptographic Failures

Erreurs Courantes

```
// MAUVAISES PRATIQUES
// 1. Algorithme faible
MessageDigest md = MessageDigest.getInstance("MD5");

// 2. Clé codée en dur
private static final String SECRET_KEY = "mySecretKey123";

// 3. Mode ECB (patterns visibles)
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

// 4. Seed prévisible
Random random = new Random(); // Pas cryptographique!
```

A02 : Bonnes Pratiques Cryptographiques

```java
// BONNES PRATIQUES
public class SecureCrypto {

    // 1. Algorithme fort
    private static final String ALGORITHM = "AES/GCM/NoPadding";

    // 2. Clé depuis un vault sécurisé
    @Value("${vault.encryption.key}")
    private String encryptionKey;

    // 3. IV aléatoire pour chaque chiffrement
    public byte[] encrypt(byte[] data) throws Exception {
        SecureRandom secureRandom = new SecureRandom();
        byte[] iv = new byte[12]; // 96 bits pour GCM
        secureRandom.nextBytes(iv);

        Cipher cipher = Cipher.getInstance(ALGORITHM);
        GCMParameterSpec spec = new GCMParameterSpec(128, iv);
        cipher.init(Cipher.ENCRYPT_MODE, getKeyFromVault(), spec);

        byte[] encrypted = cipher.doFinal(data);
        return concatenate(iv, encrypted);
    }
}
```

A07 : Identification and Authentication Failures

Vulnérabilités Courantes

- Mots de passe faibles acceptés
- Pas de protection contre brute-force
- Tokens de session prévisibles
- Pas de MFA pour les opérations sensibles

A07 : Implémentation Sécurisée

```java
@Service
public class AuthenticationService {

    private final LoadingCache<String, Integer> loginAttempts =
        CacheBuilder.newBuilder()
            .expireAfterWrite(15, TimeUnit.MINUTES)
            .build(CacheLoader.from(() -> 0));

    public AuthResult authenticate(LoginRequest request) {
        String ip = request.getIpAddress();

        // Protection brute-force
        if (loginAttempts.getIfPresent(ip) != null
                && loginAttempts.getIfPresent(ip) >= 5) {
            throw new TooManyAttemptsException("Compte temporairement bloqué");
        }

        try {
            User user = userService.findByEmail(request.getEmail());
            if (passwordService.verify(request.getPassword(), user.getHash())) {
                loginAttempts.invalidate(ip);
                return generateSecureSession(user);
            }
        } catch (Exception e) {
            // Log et incrémentation des tentatives
        }

        loginAttempts.put(ip, loginAttempts.getIfPresent(ip) + 1);
        throw new AuthenticationException("Identifiants invalides");
    }
}
```

A09 : Security Logging and Monitoring Failures

Ce qu'il faut logger

```
@Aspect
@Component
public class SecurityAuditAspect {

    private static final Logger securityLog =
        LoggerFactory.getLogger("SECURITY_AUDIT");

    @Around("@annotation(audited)")
    public Object auditSecurityEvent(ProceedingJoinPoint pjp,
                                     Audited audited) throws Throwable {
        SecurityContext ctx = SecurityContextHolder.getContext();
        String user = ctx.getAuthentication().getName();
        String action = audited.action();
        String ip = RequestContextHolder.getIpAddress();

        try {
            Object result = pjp.proceed();
            securityLog.info("SUCCESS | User: {} | Action: {} | IP: {} | " +
                "Method: {}", user, action, ip, pjp.getSignature());
            return result;
        } catch (Exception e) {
            securityLog.warn("FAILURE | User: {} | Action: {} | IP: {} | " +
                "Error: {}", user, action, ip, e.getMessage());
            throw e;
        }
    }
}
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Cadres Réglementaires

RGPD : Principes Fondamentaux

Les 7 Principes

Principe	Description	Implication Technique
Licéité	Base légale pour le traitement	Gestion des consentements
Limitation	Finalité définie et légitime	Contrôle d'usage des données
Minimisation	Données strictement nécessaires	Collecte minimale
Exactitude	Données à jour	Mécanismes de mise à jour
Conservation	Durée limitée	Purge automatique
Sécurité	Protection appropriée	Chiffrement, accès contrôlé
Responsabilité	Prouver la conformité	Audit, documentation

RGPD : Implémentation en Java

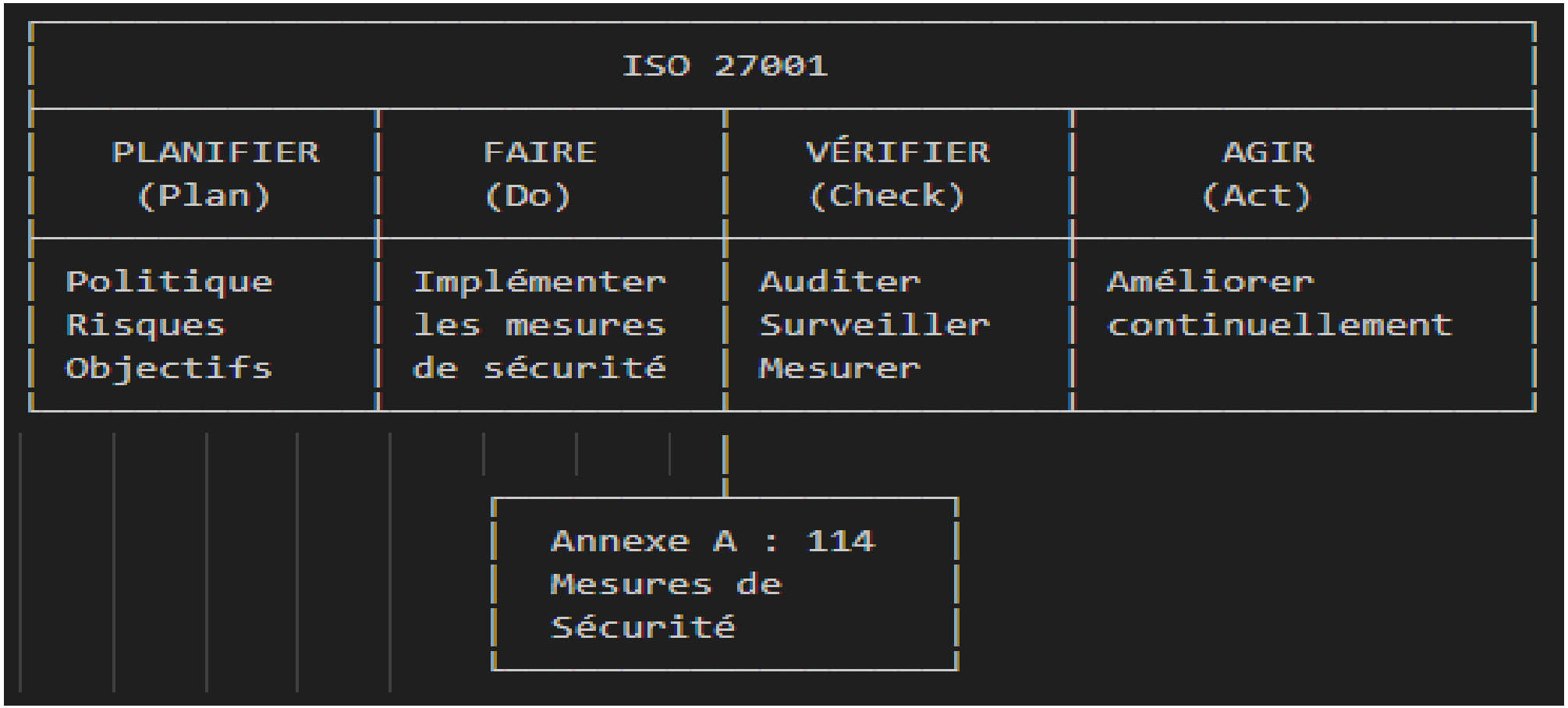
```java
@Entity
@Table(name = "users")
public class User {

    @Id
    private Long id;

    // Données personnelles chiffrées
    @Convert(converter = EncryptedStringConverter.class)
    @PersonalData(purpose = "IDENTIFICATION", retention = "5 YEARS")
    private String email;

    @Convert(converter = EncryptedStringConverter.class)
    @PersonalData(purpose = "CONTACT", retention = "5 YEARS")
    private String phone;

    // Consentements
    @OneToMany(mappedBy = "user")
    private List<Consent> consents;

    // Droit à l'oubli
    public void anonymize() {
        this.email = "anonymized_" + id + "@deleted.local";
        this.phone = null;
        this.consents.forEach(Consent::revoke);
    }
}
```

RGPD : Gestion des Consentements

```java
@Service
public class ConsentService {

    public void recordConsent(Long userId, String purpose, boolean granted) {
        Consent consent = Consent.builder()
            .userId(userId)
            .purpose(purpose)
            .granted(granted)
            .timestamp(Instant.now())
            .ipAddress(getCurrentIpAddress())
            .version(getCurrentPolicyVersion())
            .build();

        consentRepository.save(consent);

        // Audit trail
        auditService.log(AuditEvent.CONSENT_RECORDED, consent);
    }

    public boolean hasValidConsent(Long userId, String purpose) {
        return consentRepository
            .findLatestConsent(userId, purpose)
            .map(c -> c.isGranted() && !c.isExpired())
            .orElse(false);
    }

    @Scheduled(cron = "0 0 2 * * *") // Tous les jours à 2h
    public void purgeExpiredData() {
        userRepository.findUsersForDeletion()
            .forEach(user -> user.anonymize());
    }
}
```

ISO 27001 : Système de Management

Structure de la Norme



ISO 27001 : Implémentation du SMSI

```
@Configuration
public class SecurityManagementConfig {

    // A.9 - Contrôle d'accès
    @Bean
    public AccessControlPolicy accessControlPolicy() {
        return AccessControlPolicy.builder()
            .principle(LeastPrivilege.ENABLED)
            .authentication(AuthType.MFA)
            .sessionTimeout(Duration.ofMinutes(30))
            .build();
    }

    // A.10 - Cryptographie
    @Bean
    public CryptographyPolicy cryptographyPolicy() {
        return CryptographyPolicy.builder()
            .algorithm("AES-256-GCM")
            .keyRotation(Duration.ofDays(90))
            .keyManagement(KeyManagement.HSM)
            .build();
    }

    // A.12 - Sécurité des opérations
    @Bean
    public OperationsSecurityPolicy operationsPolicy() {
        return OperationsSecurityPolicy.builder()
            .changeManagement(ENABLED)
            .capacityManagement(ENABLED)
            .malwareProtection(ENABLED)
            .backup(BackupPolicy.DAILY_ENCRYPTED)
            .build();
    }
}
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Synthèse

Points Clés à Retenir

Concept	Point Essentiel
Triade CIA	Confidentialité + Intégrité + Disponibilité
Security by Design	Intégrer la sécurité dès la conception
Least Privilege	Droits minimum nécessaires
Defense in Depth	Plusieurs couches de protection
Fail Secure	En cas d'erreur, bloquer par défaut
SDLC Sécurisé	Sécurité à chaque phase du développement
OWASP Top 10	Vulnérabilités les plus critiques
RGPD	Protection des données personnelles

Checklist du Développeur Sécurisé

Conception

- [ ] Threat modeling réalisé
- [ ] Exigences de sécurité définies
- [ ] Architecture revue

Codage

- [ ] Validation des entrées
- [ ] Requêtes paramétrées
- [ ] Encodage des sorties
- [ ] Gestion sécurisée des erreurs

Authentification

- [ ] Mots de passe hashés (BCrypt)
- [ ] Protection brute-force
- [ ] Sessions sécurisées

Données

- [ ] Chiffrement au repos
- [ ] Chiffrement en transit (TLS)
- [ ] Clés gérées de manière sécurisée

Déploiement

- [ ] Headers de sécurité configurés
- [ ] Secrets dans un vault
- [ ] Monitoring activé

Ressources

Documentation

- OWASP : owasp.org
- NIST : nvd.nist.gov
- CNIL (RGPD) : cnil.fr

Outils

- SonarQube (SAST)
- OWASP ZAP (DAST)
- Dependency-Check

Formations

- SANS Secure Coding
- Certified Secure Software Lifecycle Professional (CSSLP)

Questions ?

Merci pour votre attention !