Sécurité des Applications Web - Jour 1

m2iformation.fr

Sécurité des Applications Web

Jour 1 : Introduction à la Cybersécurité et DevSecOps

Programme de la Formation

Jour	Contenu
Jour 1	Introduction cybersécurité, DevSecOps, Technologies de sécurisation
Jour 2	Vulnérabilités OWASP Top 10, Exploitation pratique
Jour 3	Outils d'attaque, Attaques avancées (LFI, RFI, RCE)
Jour 4	Sécurisation du développement, Bonnes pratiques
Jour 5	Automatisation SAST/DAST, Durcissement système

Objectifs du Jour 1

A l'issue de cette journée, vous serez capable de :

- Identifier les enjeux de sécurité d'un système d'information
- Reconnaitre les profils d'attaquants et leurs méthodes
- Comprendre la Cyber Kill Chain
- Connaitre les référentiels et normes de sécurité
- Intégrer la sécurité dans le cycle DevOps
- Maitriser les mécanismes d'authentification et de chiffrement

Module 1

Introduction à la Cybersécurité

Les Enjeux d'un Système d'Information

Définition

Un Système d'Information (SI) représente l'ensemble organisé des ressources (matériels, logiciels, personnel, données, procédures) permettant de collecter, stocker, traiter et diffuser l'information au sein d'une organisation.

Les critères de sécurité fondamentaux

La sécurité d'un SI repose sur quatre piliers essentiels, souvent désignés par l'acronyme DICP :

Les Enjeux d'un Système d'Information

- **Disponibilité** : garantir l'accès au SI quand nécessaire
- **Intégrité** : assurer que les données ne sont pas altérées
- **Confidentialité** : restreindre l'accès aux seules personnes autorisées
- **Preuve** : conserver une trace des actions effectuées

Le Panorama des Risques Actuels

Statistiques clés (2023-2024)

Indicateur	Valeur
Cout moyen d'une violation de données	4,45 millions USD
Délai moyen de détection d'une brèche	204 jours
Pourcentage d'attaques via phishing	36%
Augmentation des ransomwares	+95% par an

Secteurs les plus ciblés

Les secteurs financier, santé, industrie et administration publique concentrent la majorité des attaques en raison de la valeur des données qu'ils manipulent.

Les Profils des Attaquants

Classification par motivation

Profil	Motivation	Cibles	Moyens
Script Kiddies	Notoriété, défi	Opportunistes	Outils existants
Hacktivistes	Idéologie	Symboliques	Variables
Cybercriminels	Profit financier	Rentables	Sophistiqués
Menaces internes	Vengeance, profit	Employeur	Accès privilégiés
APT (Etats)	Espionnage	Stratégiques	Très avancés

Advanced Persistent Threat (APT)

Les APT désignent des attaques ciblées, persistantes et sophistiquées, généralement menées par des groupes étatiques ou para-étatiques.

Les Méthodes des Attaquants

Techniques d'attaque courantes

Ingénierie sociale

Manipulation psychologique visant à obtenir des informations confidentielles. Le phishing en est la forme la plus répandue.

Exploitation de vulnérabilités

Utilisation de failles logicielles connues (CVE) ou inconnues (zero-day) pour compromettre un système.

Les Méthodes des Attaquants

Attaques par force brute

Tentatives répétées et systématiques pour deviner des identifiants ou des clés de chiffrement.

Attaques de la chaine d'approvisionnement

Compromission d'un fournisseur ou d'un composant logiciel pour atteindre la cible finale.

Les Vecteurs d'Attaque

Points d'entrée d'un SI

Les vecteurs principaux sont : les applications web exposées, les emails (phishing), les accès distants mal sécurisés, et les équipements connectés.

Les Grandes Familles d'Attaques

Classification des attaques

Famille	Description	Exemples
Reconnaissance	Collecte d'informations	Scan de ports, OSINT
Intrusion	Accès non autorisé	Exploitation de failles
Elévation de privilèges	Obtention de droits supérieurs	Kernel exploits
Exfiltration	Vol de données	Tunneling DNS
Persistance	Maintien de l'accès	Backdoors, rootkits
Destruction	Sabotage	Ransomware, wiper
Déni de service	Indisponibilité	DDoS, amplification

La Cyber Kill Chain

Modèle Lockheed Martin

La Cyber Kill Chain décrit les sept phases d'une cyberattaque sophistiquée :

Phase	Description	Exemple
1. Reconnaissance	Identification de la cible	Recherche LinkedIn, DNS
2. Armement	Création de la charge malveillante	Malware + exploit
3. Livraison	Transmission à la cible	Email, site web compromis
4. Exploitation	Déclenchement de la vulnérabilité	Buffer overflow
5. Installation	Implantation du malware	Backdoor, RAT
6. Commande et Controle	Communication avec l'attaquant	C2 server
7. Actions sur objectifs	Réalisation de l'objectif	Exfiltration, destruction

Cyber Kill Chain - Schéma

Interet de la Cyber Kill Chain

Application défensive

La connaissance de ce modèle permet de :

Détecter : identifier les indicateurs de compromission (IoC) à chaque phase

Prévenir : mettre en place des controles adaptés à chaque étape

Répondre : interrompre l'attaque le plus tot possible dans la chaine

Interet de la Cyber Kill Chain

Principe de rupture

Plus l'attaque est interrompue tot, moins les dommages sont importants. L'objectif est de "briser la chaine" avant la phase d'actions sur objectifs.

Module 2

Le Pilotage et la Maitrise des Risques

Prise de Conscience au Niveau Etatique

Evolution historique

La cybersécurité est devenue une priorité nationale suite à plusieurs incidents majeurs :

- 2007 : Attaques contre l'Estonie (premier cas de cyberguerre)
- 2010 : Stuxnet (sabotage industriel iranien)
- 2017 : WannaCry et NotPetya (impacts mondiaux)

Prise de Conscience au Niveau Etatique

En France

La création de l'ANSSI en 2009 marque la structuration de la cyberdéfense nationale. La Loi de Programmation Militaire (LPM) impose depuis 2013 des obligations aux Opérateurs d'Importance Vitale (OIV).

Organismes de Référence

Organismes étatiques francais

Organisme	Role
ANSSI	Autorité nationale en matière de sécurité des SI
SGDSN	Coordination interministérielle de la défense et sécurité
CNIL	Protection des données personnelles
CERT-FR	Réponse aux incidents de sécurité informatique

Organismes internationaux

Organisme	Role
ENISA	Agence européenne de cybersécurité
NIST	Standards américains de sécurité
OWASP	Sécurité des applications web (communauté)
MITRE	Base de connaissances ATT&CK

Exigences Légales et Contexte Juridique

Cadre pénal francais

Article 323-1 du Code Pénal (Loi Godfrain, 1988)

Le fait d'accéder ou de se maintenir frauduleusement dans tout ou partie d'un système de traitement automatisé de données est puni de :

- 2 ans d'emprisonnement et 60 000 euros d'amende
- 3 ans et 100 000 euros si modification ou suppression de données
- 5 ans et 150 000 euros si atteinte à l'Etat

Exigences Légales et Contexte Juridique

Règlement Général sur la Protection des Données (RGPD)

Applicable depuis mai 2018, le RGPD impose :

- Notification des violations sous 72 heures
- Sanctions jusqu'à 20 millions d'euros ou 4% du CA mondial
- Obligation de sécurisation des traitements (article 32)

Les Référentiels et Normes

Normes internationales

Norme	Domaine	Description
ISO 27001	SMSI	Système de management de la sécurité de l'information
ISO 27002	Bonnes pratiques	Mesures de sécurité recommandées
ISO 27005	Risques	Gestion des risques liés à la sécurité
IEC 62443	Industriel	Sécurité des systèmes industriels

Référentiels sectoriels

Référentiel	Secteur	Obligation
PCI-DSS	Paiement	Obligatoire pour traitement cartes bancaires
HDS	Santé	Hébergement de données de santé en France
SOC 2	Services	Audit des controles de sécurité

Standards de Gestion des Vulnérabilités

Identification des vulnérabilités

Standard	Description
CVE	Common Vulnerabilities and Exposures - Identifiant unique
CWE	Common Weakness Enumeration - Classification des faiblesses
NVD	National Vulnerability Database - Base NIST
CVSS	Common Vulnerability Scoring System - Score de gravité

Standards de Gestion des Vulnérabilités

Score CVSS

Le CVSS évalue la gravité d'une vulnérabilité sur une échelle de 0 à 10 :

Score	Gravité	Exemple
0.0 - 3.9	Faible	Information disclosure mineur
4.0 - 6.9	Moyen	XSS stocké
7.0 - 8.9	Elevé	Injection SQL
9.0 - 10.0	Critique	RCE sans authentification

Le Framework MITRE ATT&CK

Présentation

ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) est une base de connaissances des tactiques et techniques utilisées par les attaquants.

Structure de la matrice

Tactique	Objectif	Exemples de techniques
Reconnaissance	Collecter des informations	Scan actif, Phishing
Accès initial	Pénétrer le réseau	Exploit public, Drive-by
Exécution	Lancer du code malveillant	PowerShell, Scripts
Persistance	Maintenir l'accès	Taches planifiées, Services
Elévation	Obtenir plus de droits	Exploitation kernel
Exfiltration	Voler les données	Canal C2, Protocole alternatif

Threat Modeling

Définition

Le Threat Modeling (modélisation des menaces) est une approche structurée pour identifier les menaces potentielles et définir les contre-mesures appropriées.

Méthodologie STRIDE

Menace	Description	Propriété violée
Spoofing	Usurpation d'identité	Authentification
Tampering	Modification de données	Intégrité
Repudiation	Nier une action	Non-répudiation
Information Disclosure	Fuite d'information	Confidentialité
Denial of Service	Rendre indisponible	Disponibilité
Elevation of Privilege	Obtenir plus de droits	Autorisation

Enjeux de la Sécurité des Applications Web

Pourquoi les applications web sont-elles ciblées

Exposition permanente

Les applications web sont accessibles 24h/24 depuis Internet, offrant une surface d'attaque constante.

Complexité croissante

L'architecture moderne (microservices, API, SPA) multiplie les points de vulnérabilité.

Enjeux de la Sécurité des Applications Web

Pourquoi les applications web sont-elles ciblées

Données sensibles

Les applications web manipulent souvent des données personnelles, financières ou stratégiques.

Développement rapide

La pression sur les délais de mise en production peut conduire à négliger la sécurité.

Module 3

Introduction au DevSecOps

Le Role de la Sécurité dans le Développement

Evolution des approches

Approche traditionnelle (Waterfall)

La sécurité intervenait en fin de projet, souvent trop tard pour corriger les problèmes structurels.

Approche Agile

L'intégration continue a accéléré les livraisons mais la sécurité restait en retrait.

Le Role de la Sécurité dans le Développement

Approche DevSecOps

La sécurité est intégrée à chaque étape du cycle de développement, de la conception au déploiement.

Principe fondamental

"Shift Left Security" : déplacer les controles de sécurité le plus tot possible dans le cycle de développement.

La Méthodologie DevSecOps

Le cycle DevSecOps

Les Principes de Sécurité dans le Développement

Secure by Design

Principe du moindre privilège

Accorder uniquement les droits strictement nécessaires à chaque composant.

Défense en profondeur

Multiplier les couches de sécurité pour qu'une faille unique ne compromette pas tout le système.

Fail Secure

En cas d'erreur, le système doit échouer de manière sécurisée (fermer l'accès plutot que l'ouvrir).

Les Outils DevSecOps

Chaine d'intégration continue

Outil	Fonction	Usage
GitLab CI/CD	Pipeline complet	Orchestration des jobs
Jenkins	Automatisation	Builds et déploiements
SonarQube	Qualité de code	Analyse statique
Trivy	Scan de conteneurs	Vulnérabilités images
OWASP ZAP	Test dynamique	Scan d'applications
HashiCorp Vault	Gestion des secrets	Credentials sécurisés

Exemple de Pipeline DevSecOps

Le code ci-dessous illustre un pipeline GitLab CI/CD intégrant des controles de sécurité. Chaque stage représente une étape du processus.

```
# Fichier .gitlab-ci.yml
# Ce fichier définit les étapes du pipeline CI/CD

stages:
  - build      # Compilation de l'application
  - security   # Tests de sécurité
  - test       # Tests fonctionnels
  - deploy     # Déploiement

# Stage de compilation
build:
  stage: build
  script:
    - npm install    # Installation des dépendances
    - npm run build  # Compilation du code source
```

Pipeline DevSecOps (Suite)

```
# Analyse statique du code source (SAST)
# Détecte les vulnérabilités dans le code avant exécution
sast:
  stage: security
  script:
    # SonarQube analyse le code et remonte les problèmes
    # -Dsonar.projectKey identifie le projet
    # -Dsonar.sources indique le répertoire à analyser
    - sonar-scanner -Dsonar.projectKey=mon-projet
                    -Dsonar.sources=src/

# Scan des dépendances pour détecter les composants vulnérables
dependency_check:
  stage: security
  script:
    # npm audit vérifie les vulnérabilités connues (CVE)
    # des packages npm utilisés dans le projet
    - npm audit --audit-level=high
```

Pipeline DevSecOps (Fin)

```yaml
# Analyse dynamique (DAST) sur l'application déployée
# Simule des attaques sur l'application en cours d'exécution
dast:
  stage: security
  script:
    # OWASP ZAP effectue un scan automatisé
    # -t spécifie l'URL cible à tester
    # -r génère un rapport au format HTML
    - zap-baseline.py -t https://staging.example.com
                      -r rapport-zap.html

# Déploiement uniquement si tous les tests passent
deploy:
  stage: deploy
  script:
    - kubectl apply -f kubernetes/  # Déploiement Kubernetes
  only:
    - main  # Uniquement sur la branche principale
```

Les Frameworks Sécurisés

Backend

Framework	Langage	Fonctionnalités de sécurité
Spring Security	Java	Authentification, autorisation, CSRF
Django	Python	ORM anti-injection, CSRF, XSS
Express + Helmet	Node.js	Headers de sécurité HTTP
ASP.NET Core	C#	Identity, anti-forgery, CORS

Frontend

Framework	Protection
React	Echappement automatique JSX
Angular	Sanitization, CSP strict
Vue.js	Echappement des templates

SAST et DAST

Analyse Statique (SAST - Static Application Security Testing)

L'analyse statique examine le code source sans l'exécuter :

- Détection précoce des vulnérabilités
- Couverture complète du code
- Intégration dans l'IDE possible
- Limitation : faux positifs fréquents

SAST et DAST

Analyse Dynamique (DAST - Dynamic Application Security Testing)

L'analyse dynamique teste l'application en cours d'exécution :

- Détection des vulnérabilités réelles
- Test en conditions proches de la production
- Pas besoin du code source
- Limitation : couverture partielle

Module 4

Rappels sur les Technologies du Web

Le Protocole HTTP

Fonctionnement

HTTP (HyperText Transfer Protocol) est le protocole de communication entre un client (navigateur) et un serveur web.

```
CLIENT                          SERVEUR
   |                               |
   |  -------- Requete HTTP -------->  |
   |  GET /page.html HTTP/1.1          |
   |  Host: www.exemple.com            |
   |                                   |
   |  <------- Réponse HTTP --------   |
   |  HTTP/1.1 200 OK                  |
   |  Content-Type: text/html          |
   |  <html>...</html>                 |
   |                                   |
```

Les Méthodes HTTP

Méthodes principales

Méthode	Usage	Corps	Idempotent
GET	Récupérer une ressource	Non	Oui
POST	Créer une ressource	Oui	Non
PUT	Remplacer une ressource	Oui	Oui
PATCH	Modifier partiellement	Oui	Non
DELETE	Supprimer une ressource	Non	Oui
HEAD	Récupérer les en-tetes	Non	Oui
OPTIONS	Connaitre les méthodes autorisées	Non	Oui

Une méthode est idempotente si des appels répétés produisent le meme résultat.

Les Headers HTTP

Headers de requete importants

```
GET /api/users HTTP/1.1
Host: api.exemple.com           # Serveur cible
User-Agent: Mozilla/5.0...      # Identifiant du client
Accept: application/json        # Format de réponse souhaité
Authorization: Bearer eyJ...    # Jeton d'authentification
Cookie: session=abc123          # Cookies de session
Content-Type: application/json  # Format des données envoyées
Origin: https://app.exemple.com # Origine de la requete (CORS)
```

Ces en-tetes permettent au serveur de comprendre le contexte de la requete et d'adapter sa réponse.

Headers HTTP de Réponse

Headers de sécurité essentiels

```
HTTP/1.1 200 OK
Content-Type: application/json
# En-tetes de sécurité recommandés :

# Empêche l'interprétation MIME incorrecte
X-Content-Type-Options: nosniff

# Bloque le chargement dans une iframe (anti-clickjacking)
X-Frame-Options: DENY

# Active le filtre XSS du navigateur
X-XSS-Protection: 1; mode=block

# Force l'utilisation de HTTPS
Strict-Transport-Security: max-age=31536000; includeSubDomains

# Définit les sources de contenu autorisées
Content-Security-Policy: default-src 'self'
```

Les Status Codes HTTP

Catégories de codes de retour

Plage	Catégorie	Signification
1xx	Information	Requete recue, traitement en cours
2xx	Succès	Requete traitée avec succès
3xx	Redirection	Action supplémentaire nécessaire
4xx	Erreur client	Erreur dans la requete
5xx	Erreur serveur	Erreur coté serveur

Codes fréquents

200 OK, 201 Created, 301 Moved Permanently, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error

Travaux Pratiques - Analyse HTTP

Objectif

Analyser les requetes HTTP avec les DevTools du navigateur.

Procédure

1. Ouvrir les DevTools (F12) et l'onglet Network
2. Naviguer sur un site web
3. Observer les requetes GET générées
4. Soumettre un formulaire et observer la requete POST
5. Identifier les headers de sécurité dans les réponses

Module 5

Les Technologies de Sécurisation

Techniques d'Authentification

Evolution des mécanismes

Méthode	Principe	Sécurité
Basic Auth	Login/password en Base64	Faible
Digest Auth	Hash du password	Moyenne
NTLM	Challenge-response Microsoft	Moyenne
Kerberos	Tickets signés	Elevée
LDAP	Annuaire centralisé	Variable
MFA	Plusieurs facteurs	Elevée

Authentification Multi-Facteurs (MFA)

Combine plusieurs éléments parmi : ce que l'on sait (password), ce que l'on possède (token), ce que l'on est (biométrie).

Authentification Centralisée

Single Sign-On (SSO)

Le SSO permet à un utilisateur de s'authentifier une seule fois pour accéder à plusieurs applications.

Protocoles SSO

Protocole	Usage	Principe
CAS	Universités, entreprises	Tickets de service
SAML	Entreprise, fédération	Assertions XML signées
OAuth 2.0	APIs, applications tierces	Tokens d'accès délégués
OpenID Connect	Grand public, moderne	OAuth 2.0 + identité

OAuth 2.0 - Fonctionnement

Flux Authorization Code

Ce flux sépare l'authentification de l'autorisation et évite l'exposition des credentials.

Techniques de Hachage

Définition

Une fonction de hachage transforme une donnée de taille variable en une empreinte de taille fixe. Cette transformation est unidirectionnelle.

Algorithmes courants

Algorithme	Taille	Usage	Statut
MD5	128 bits	Historique	Obsolète
SHA-1	160 bits	Signatures anciennes	Déprécié
SHA-256	256 bits	Signatures, intégrité	Recommandé
SHA-3	Variable	Applications critiques	Recommandé
bcrypt	Variable	Mots de passe	Recommandé
Argon2	Variable	Mots de passe	Recommandé

Hachage de Mots de Passe

Pourquoi ne pas utiliser SHA-256 seul

Le code suivant illustre un hachage simple avec SHA-256 :

```
import hashlib

# Hachage simple - INSUFFISANT pour les mots de passe
password = "MonMotDePasse123"
# encode() convertit la chaine en bytes
# hexdigest() retourne le hash en hexadécimal
hash_simple = hashlib.sha256(password.encode()).hexdigest()

# Problème : deux utilisateurs avec le meme password
# auront le meme hash -> attaque par rainbow table
```

SHA-256 est trop rapide, permettant des milliards de tentatives par seconde lors d'une attaque par force brute.

Hachage Sécurisé avec bcrypt

Le code suivant montre l'utilisation correcte de bcrypt :

```
import bcrypt

# bcrypt intègre automatiquement un sel (salt) aléatoire
# et un facteur de cout (work factor) qui ralentit le calcul

password = "MonMotDePasse123"

# Hachage avec bcrypt
# gensalt() génère un sel aléatoire
# Le paramètre rounds (défaut=12) définit le cout computationnel
# Plus rounds est élevé, plus le hachage est lent
salt = bcrypt.gensalt(rounds=12)
hash_securise = bcrypt.hashpw(password.encode(), salt)

# Vérification ultérieure
# bcrypt extrait le sel du hash stocké pour la comparaison
mot_de_passe_saisi = "MonMotDePasse123"
if bcrypt.checkpw(mot_de_passe_saisi.encode(), hash_securise):
    print("Authentification réussie")
```

Techniques de Chiffrement

Chiffrement symétrique

La meme clé sert au chiffrement et au déchiffrement.

Algorithme	Taille de clé	Usage
AES-128	128 bits	Usage général
AES-256	256 bits	Données sensibles
ChaCha20	256 bits	Mobile, streaming

Techniques de Chiffrement

Chiffrement asymétrique

Deux clés distinctes : publique (chiffrement) et privée (déchiffrement).

Algorithme	Usage
RSA	Signatures, échange de clés
ECDSA	Signatures compactes
Ed25519	Signatures modernes

Exemple de Chiffrement AES

```python
from cryptography.fernet import Fernet

# Fernet utilise AES-128 en mode CBC avec HMAC pour l'intégrité
# C'est une implémentation "batteries included" sécurisée

# Génération d'une clé secrète
# Cette clé doit etre stockée de manière sécurisée (vault, HSM)
cle = Fernet.generate_key()
cipher = Fernet(cle)

# Chiffrement
# Les données sont d'abord converties en bytes
message_original = "Données confidentielles"
message_chiffre = cipher.encrypt(message_original.encode())
# Le résultat est un blob binaire illisible

# Déchiffrement
# Nécessite la meme clé utilisée pour le chiffrement
message_dechiffre = cipher.decrypt(message_chiffre).decode()
# decode() reconvertit les bytes en chaine de caractères
```

TLS (Transport Layer Security)

Fonctionnement

TLS sécurise les communications réseau en assurant :

- **Confidentialité** : chiffrement des données échangées
- **Intégrité** : détection des modifications
- **Authentification** : vérification de l'identité du serveur

Handshake TLS simplifié

```
CLIENT                              SERVEUR
   |-- ClientHello (versions, ciphers) -->|
   |<-- ServerHello + Certificat ---------|
   |-- Vérification certificat            |
   |-- Echange de clé (PreMasterSecret) -->|
   |<-- Confirmation ---------------------|
   |======= Communication chiffrée =======|
```

Certificats Numériques

Structure d'un certificat X.509

Un certificat numérique lie une identité à une clé publique :

Champ	Description
Subject	Identité du propriétaire (CN, O, OU)
Issuer	Autorité de certification émettrice
Public Key	Clé publique du propriétaire
Validity	Dates de début et fin de validité
Serial Number	Identifiant unique
Signature	Signature de l'AC sur le certificat

Chaine de confiance

Certificat du site -> Certificat intermédiaire -> Certificat racine (préinstallé)

Modèles d'Autorisation

RBAC (Role-Based Access Control)

Le controle d'accès basé sur les roles attribue des permissions à des roles, puis assigne les roles aux utilisateurs.

RBAC - Implémentation

```python
# Définition des roles et permissions
roles = {
    "admin": ["create_user", "delete_user", "read_all", "write_all"],
    "editor": ["read_all", "write_content", "publish"],
    "viewer": ["read_public"]
}

# Attribution des roles aux utilisateurs
users = {
    "alice": ["admin"],
    "bob": ["editor"],
    "charlie": ["viewer", "editor"]  # Un user peut avoir plusieurs roles
}

def has_permission(username, permission):
    """Vérifie si un utilisateur possède une permission donnée"""
    user_roles = users.get(username, [])
    for role in user_roles:
        if permission in roles.get(role, []):
            return True
    return False

# has_permission("bob", "publish") retourne True
# has_permission("charlie", "delete_user") retourne False
```

Travaux Pratiques - Environnement

Objectif

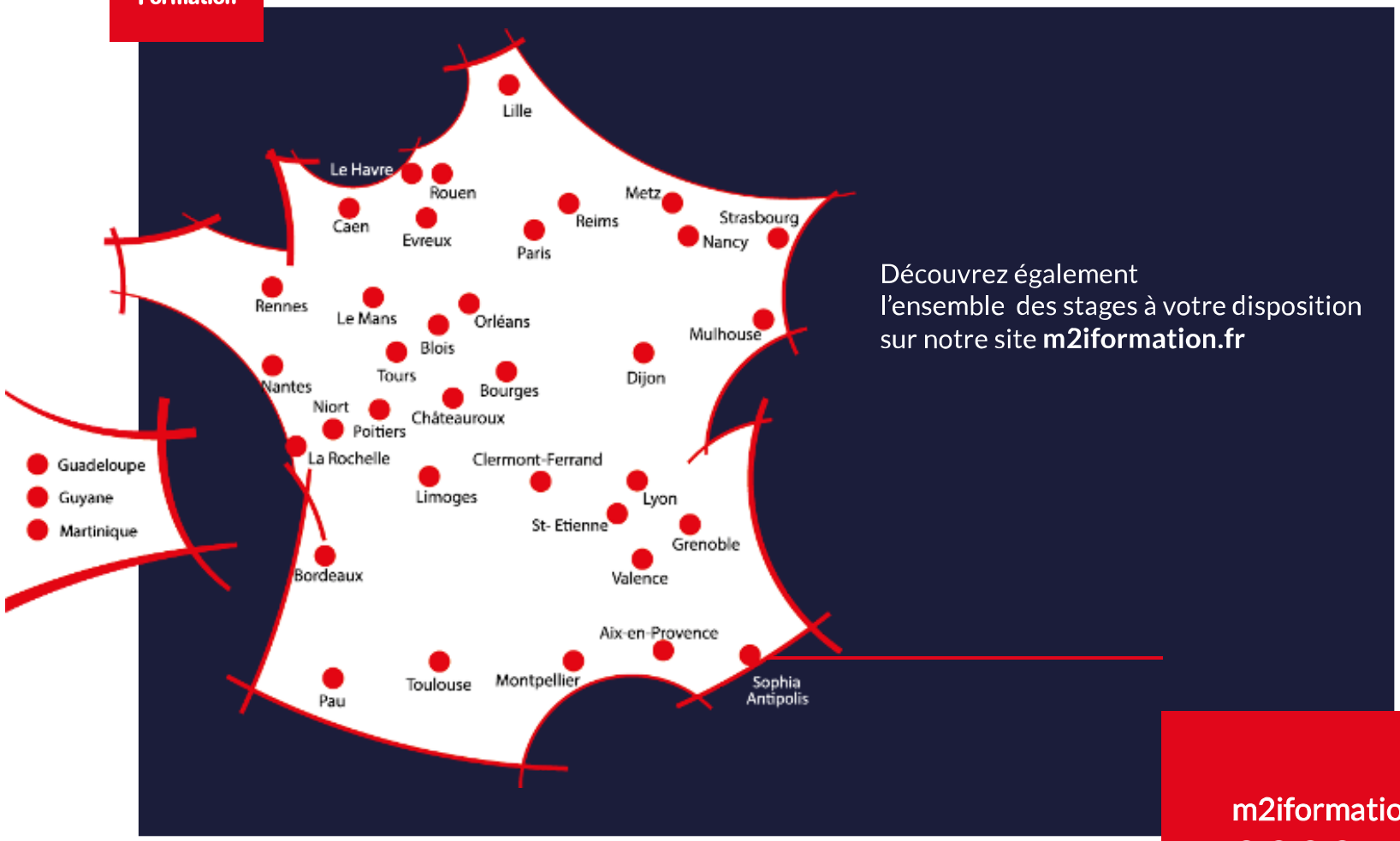
Mettre en place l'environnement de formation contenant :

- Un serveur web vulnérable
- Une base de données
- Des scripts d'exemple

Architecture

```
[Docker Host]
    |
    +-- [Container Web]     Port 8080
    |       nginx + PHP
    |
    +-- [Container DB]      Port 3306
    |       MySQL
    |
    +-- [Container Tools]
            Outils de test
```

Questions



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr