Sécurité Applicative

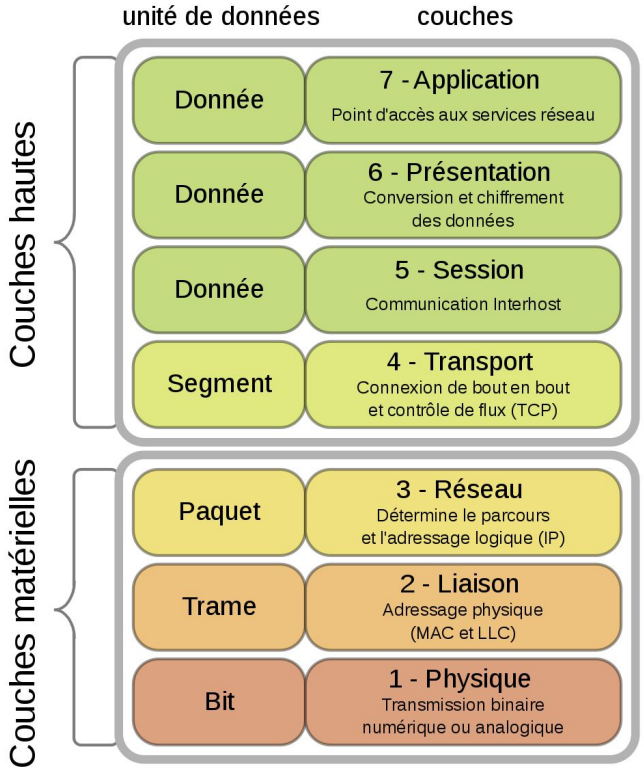
Architecture

Lu. 15 Oct. 2018 - PHELIZOT Yvan

Rappels

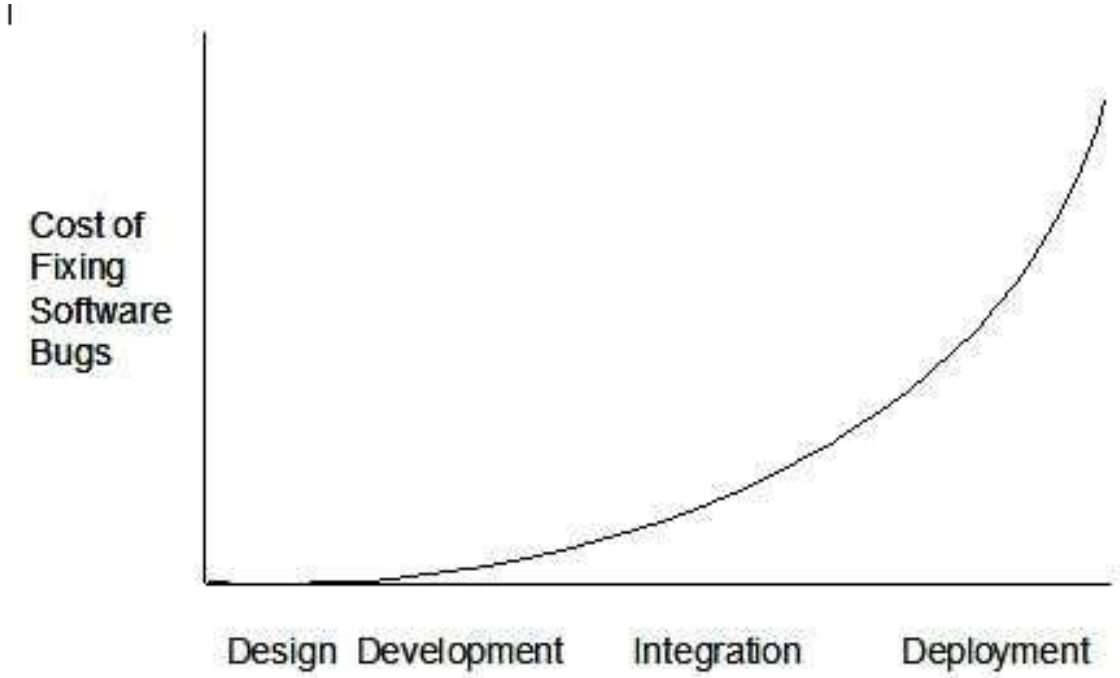
Modèle OSI

- Modèle en 7 couches
- Modèle facile à comprendre
- Communique avec les couches directement à portée



Source: https://fr.wikipedia.org/wiki/Mod%C3%A8le_OSI

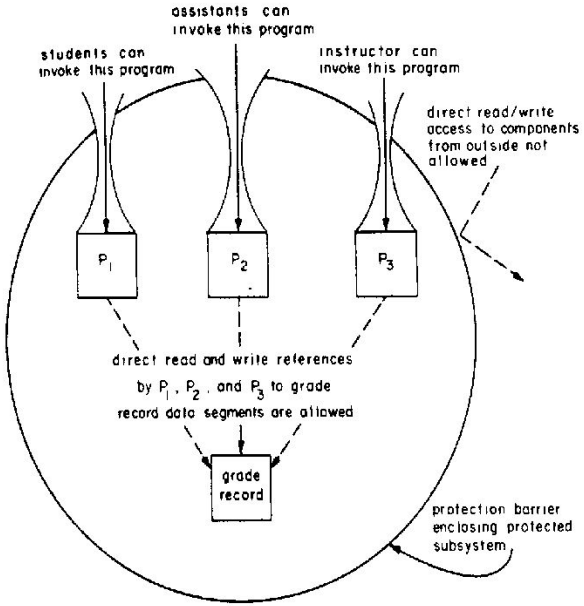
Coût d’un bug


Cost of
Fixing
Software
Bugs
Design
Development
Integration
Deployment

Principes
de conception

Saltzer and Schroeder's design principles (1975)

- Economy of mechanism
- Fail-safe defaults (and fast)
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Work factor
- Compromise recording



Saltzer and Kaashoek design principles (2009)

- Minimize secrets
- Adopt sweeping simplifications
- Least astonishment
- Design for iteration

D’autres principes importants

- Defense in depth
- Deny by default ⇒ Plan on failure
- Isolation

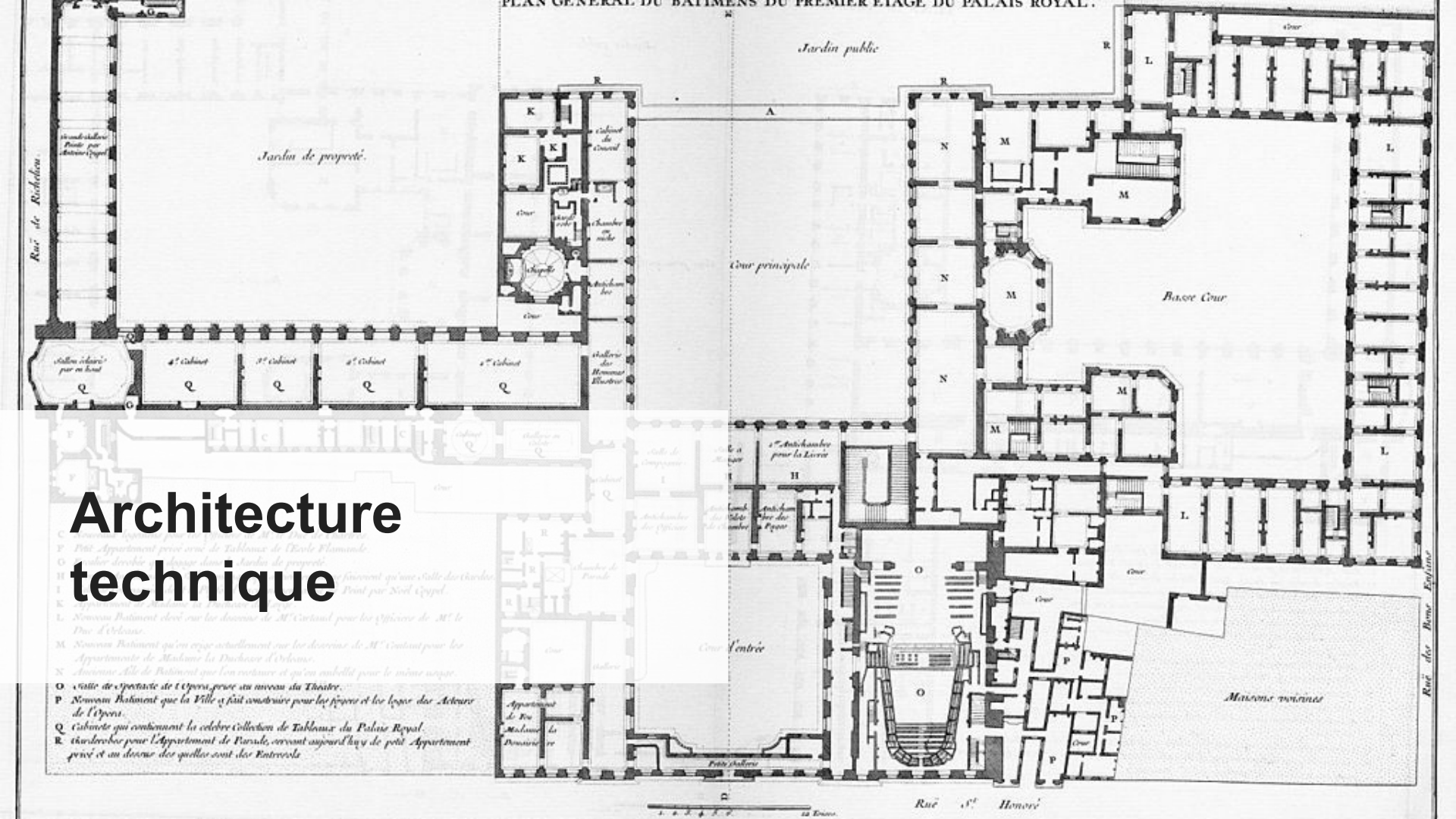
Pourquoi?

- Peu nombreux
- Simple
- Principes de haut-niveau
- Problèmes fondamentaux
- Indépendant de l’implémentation
- “Fractale”

Architecture?

Architecture

- Etymologie : principe + couvrir/clore ⇒ définition des règles de construction d’un ouvrage
- Informatique
  - Relation entre les composants d’une application
  - Souvent dur à changer une fois décidé
  - Architecture Technique
  - Architecture Logicielle
  - Architecture Fonctionnelle



Architecture technique

Architecture technique

- Composant matériel supportant l’application
  - Machines
  - Bases de données
  - Flux
  - Produits Applicatifs (OS, Serveurs, Frameworks, …)
  - Réseaux & topologie
  - Disques (SAN, NAS, …)
  - Services (API)

Composants techniques pour la sécurité

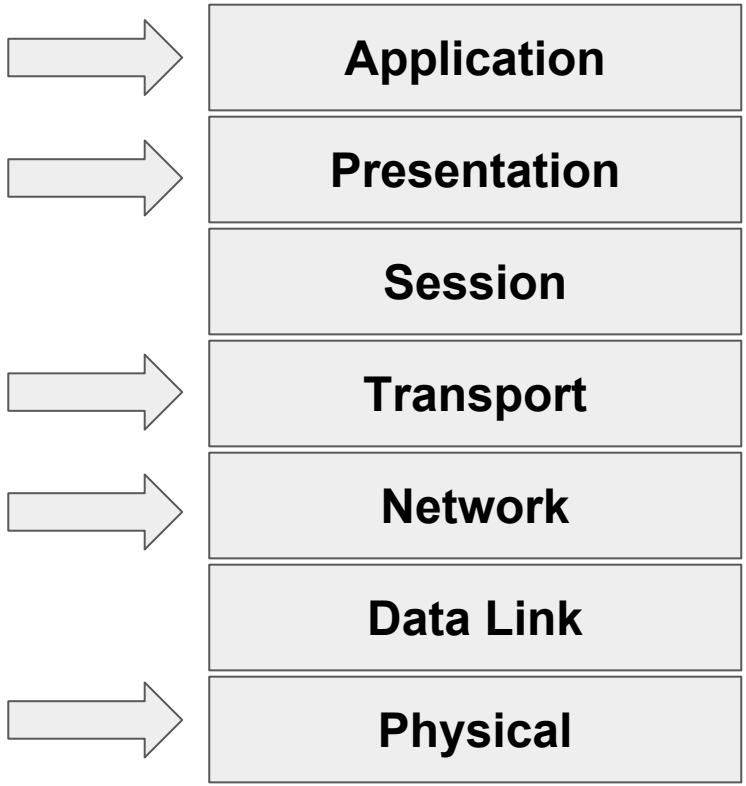
DNSSEC, SSO

WAF, IDS, IPS

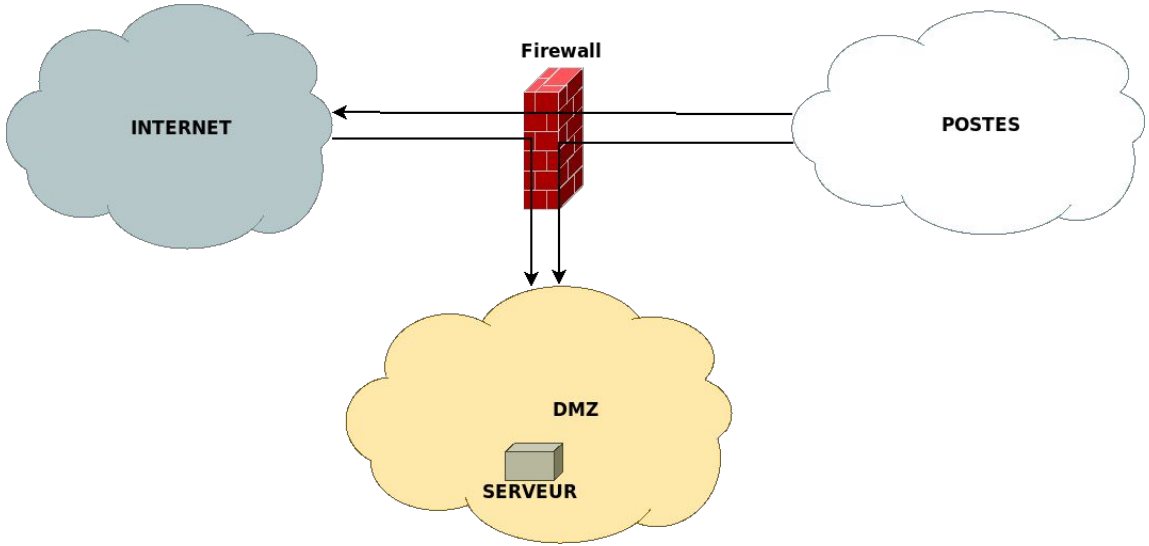
Firewall

Segmentation réseau, VLAN, IP filtering, IPSEC, VPN

Garde à l’accueil, Protection électromagnétique, WiFi

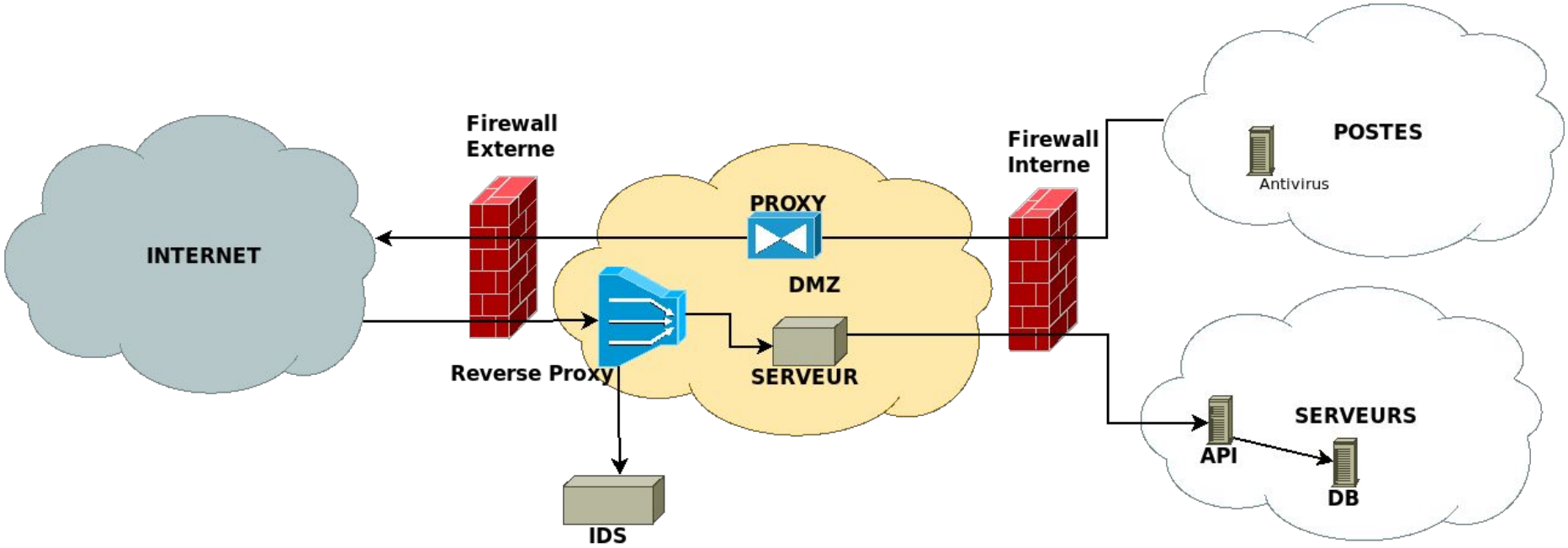


Exemples d’architecture



Architecture “Accès via pare-feu”

Exemples d’architecture



Architecture basée sur deux pare-feux

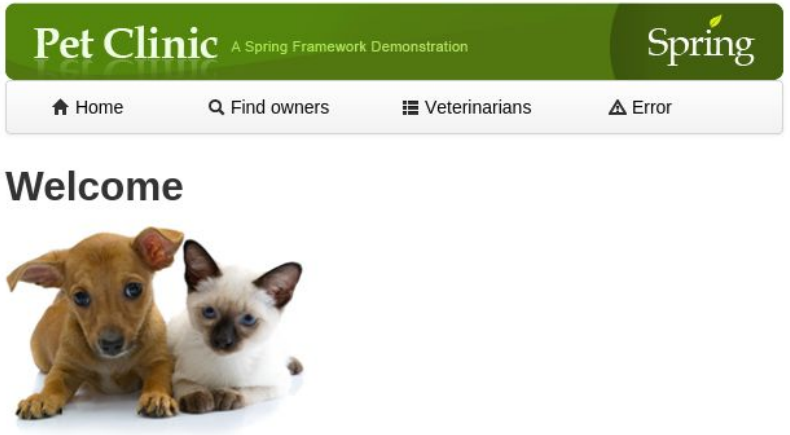
Application des principes

- Defense in Depth : 2 firewalls, Hardened systems (AppArmor, SELinux, services désactivés, ...)
- Isolation & segregation of duties : plusieurs réseaux & VLAN par fonction
- Least privilege : flux réseaux limités
- Psychological acceptability : quasi-transparent
- Defense in Depth: Systèmes à jour?
- Compromise recording: supervision
- Least common mechanism: séparation des réseaux

Pet Clinic

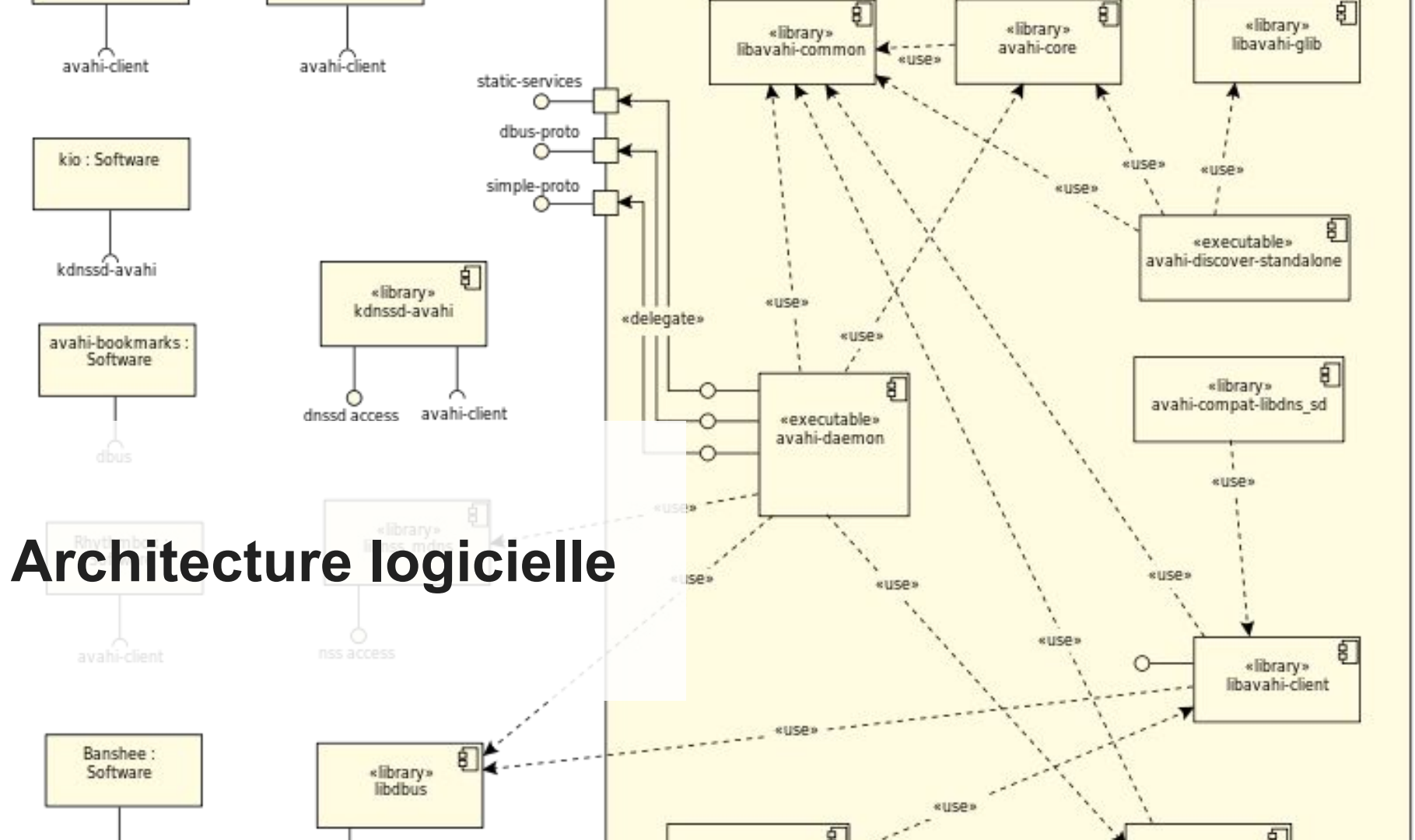
Revenons à notre clinique

Des idées d’architecture?



spring by Pivotal

Architecture logicielle



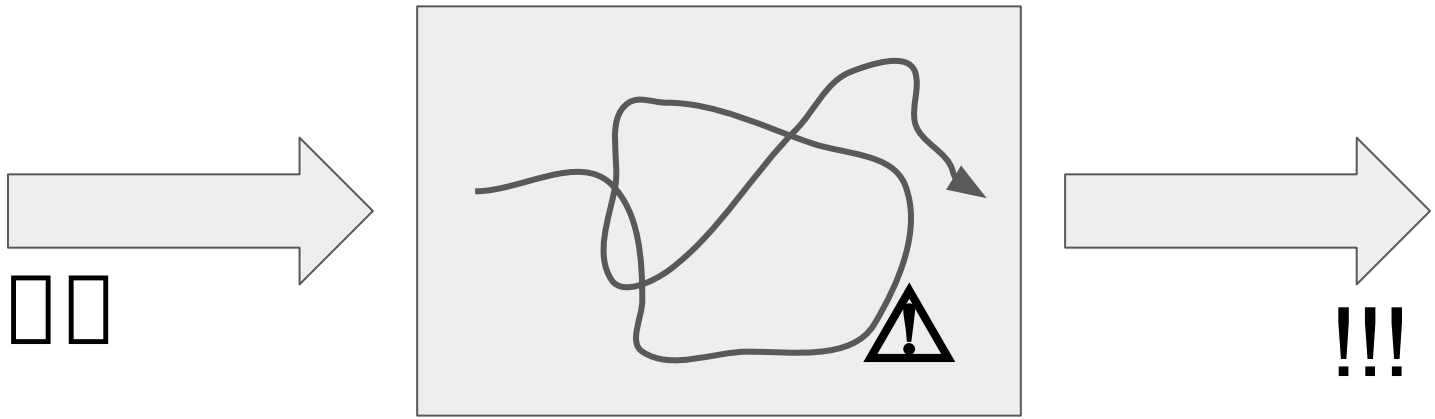
Architecture logicielle

- Description des composants logiciels constituant l’application
- Décrit comment réaliser les exigences fonctionnelles et non-fonctionnelles
- Relations entre composants (interfaces, flux, responsabilité, …)

1, 2, 3, n-tiers : architecture en couche

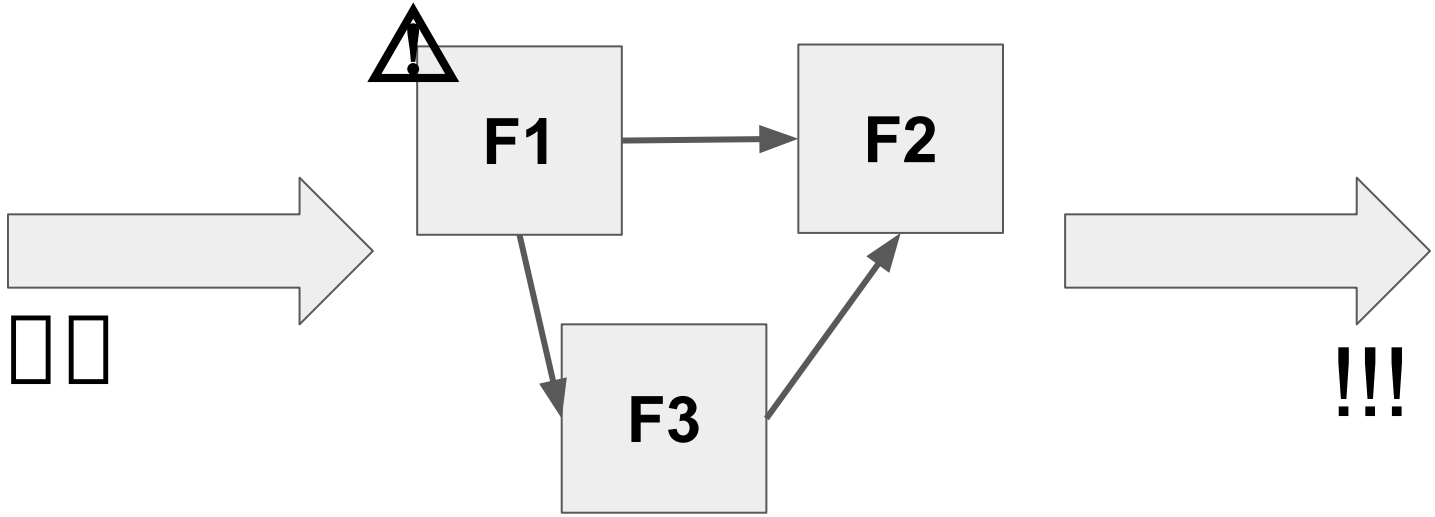
- Couche de présentation
- Couche de traitement
- Couche d’accès aux données

Architecture logicielle



Monolithe

Architecture logicielle



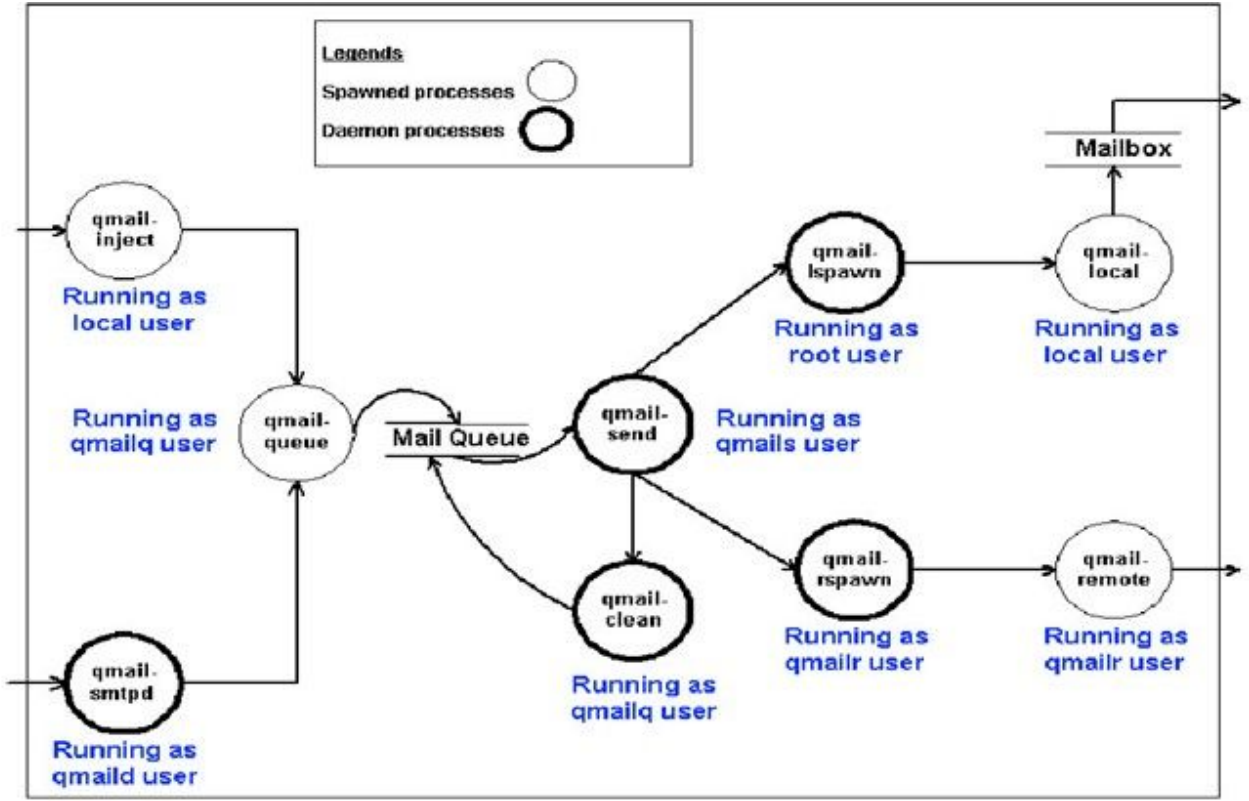
Secure Design Patterns

- Modèles existants pour résoudre des problèmes connus
- Niveau
  - Architecture: haut niveau de responsabilités entre les composants
  - Conception: décrit les interactions entre les composant
  - Implémentation: bas-niveau/secure coding

Architecture: Distrustful Decomposition

- Sépare les fonctions entre plusieurs programmes non sûres
- But: réduire la surface d’attaque. Si un bloc est exposé, l’attaquant a moins de pouvoir
- Least privilege: chaque programme s’exécute avec peu de droits
- Segregation of duties: un programme n’a pas tous les pouvoirs
- Applicable pour des programmes qui doivent gérer des données de beaucoup de façons différentes

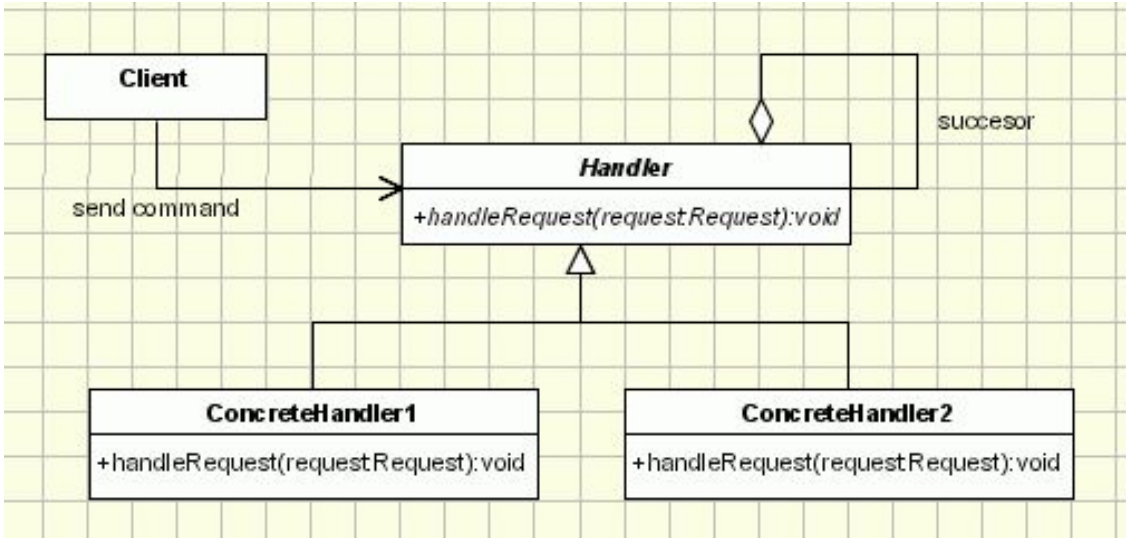
Architecture: Distrustful Decomposition



Exemples d’architecture

- Economy of mechanism : microservices
- Isolation : différents processus
- Least privilege : utilisateur particulier, droits en lecture/écriture minimaux
- Defense in depth : revalidation des composants
- Least astonishment : réutilisation de composants validés

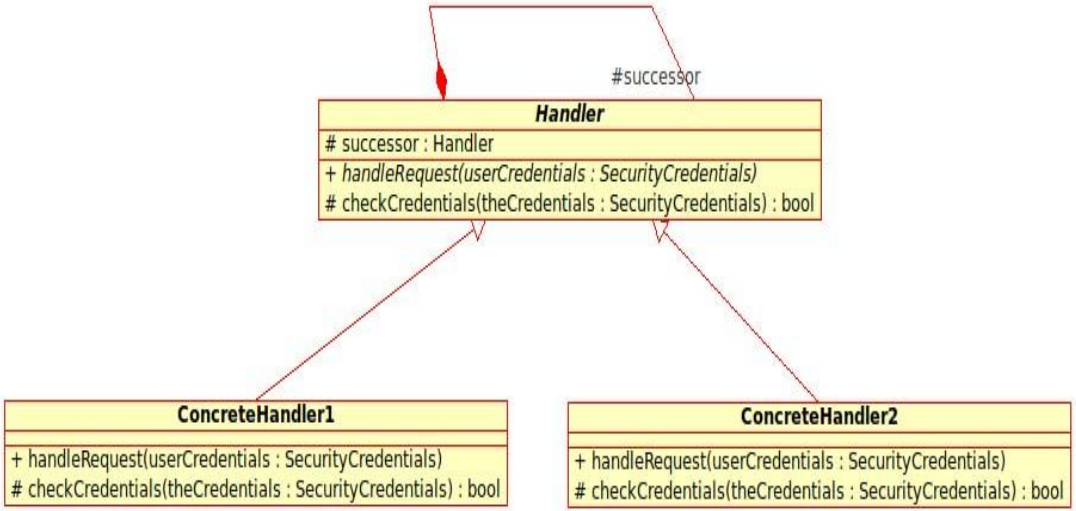
Conception: Chain of Responsibility

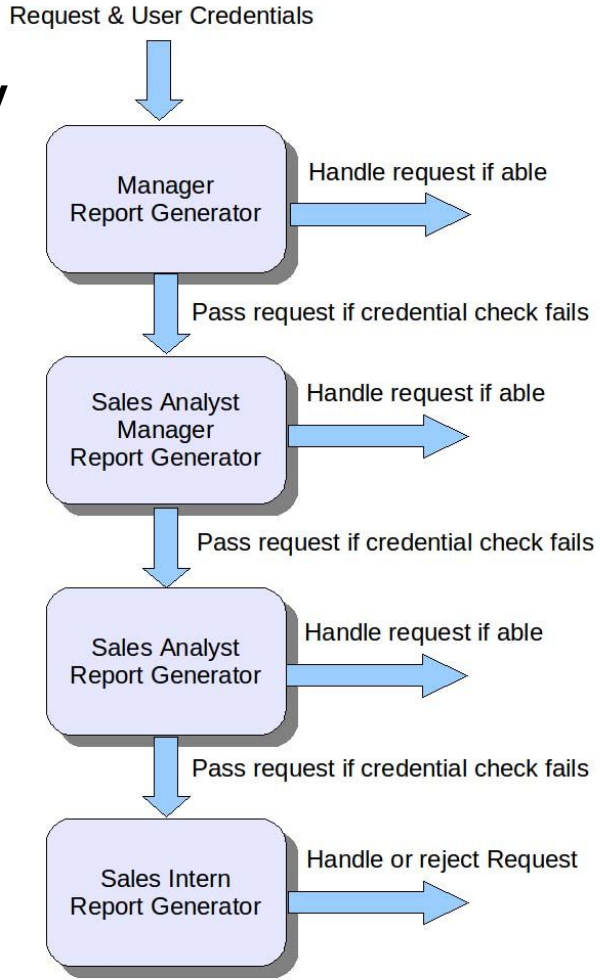


Conception: Secure Chain of Responsibility

- “Diviser pour régner”: décompose le problème en petits blocs
- Exécuter une fonction avec le maximum de droits
- Diviser la logique monolithique d’autorisation qui détermine si une fonctionnalité est accessible à un utilisateur

Conception: Chain of Responsibility





Implementation: Clear Sensitve Information

- Certains contextes peuvent entraîner une réutilisation de ressources: Mémoire, Fichier, Objet (ex: mot de passe), ...
- Risque : A3:2017 Sensitive Data Exposure
  - La mémoire peut contenir d'anciennes valeurs, le fichier peut être relu, l'objet exposé dans un log, …
- Supprimer après usage!

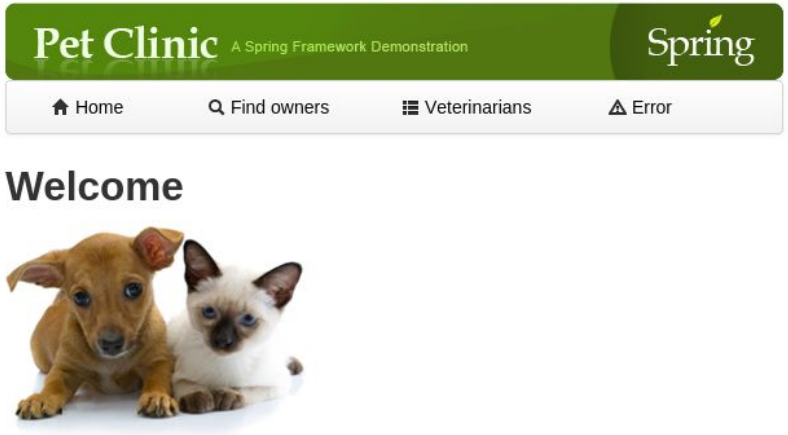
Développement

- Secure Coding
- Code Review
- Qualité de code & Bonnes pratiques de développement
- Maîtrise des outils (langages, frameworks, plateformes, …)

Pet Clinic

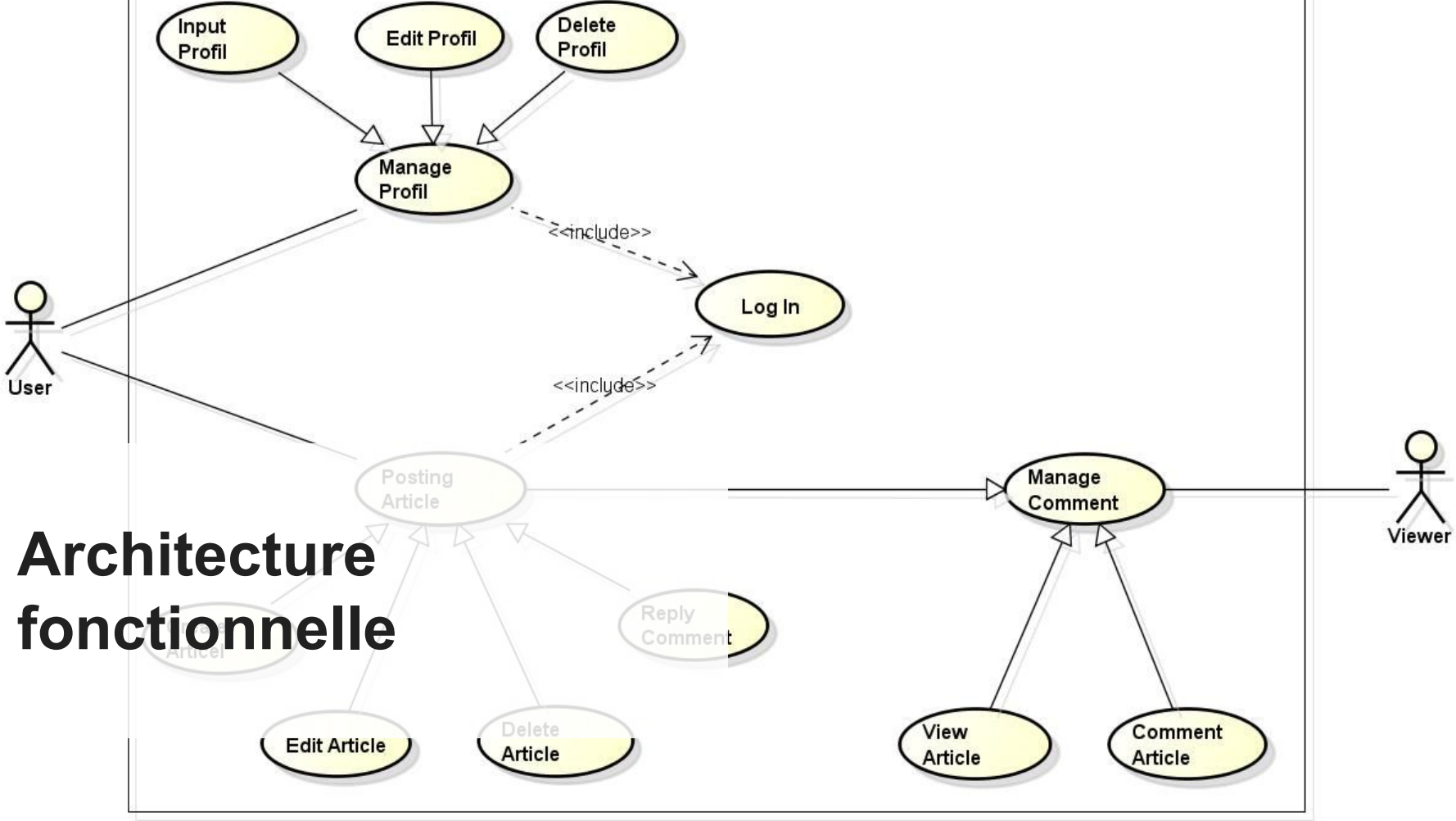
Revenons à notre clinique

- Java 7
- Spring 4
- Développé par un stagiaire



spring by Pivotal

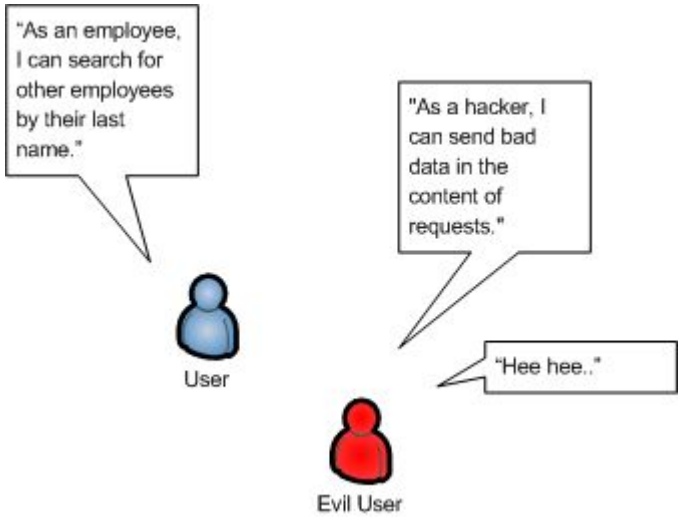
Architecture fonctionnelle



Architecture fonctionnelle

- Architecture ~ Analyse
- Décrit ce que doit faire l’application
- Description haut-niveau de la réalisation des besoins métier
- Expression du besoin, Exigences, Conception, Spécifications fonctionnelles

Sécurité?



- Exigences de sécurité descriptives ou/et prescriptives
- Prise en compte de la sécurité dans la conception (Evil/Abuser US)
- Failles introduites à la conception
  - Fonctionnalités inutiles ou dangereuses, …
  - Mauvaises pratiques

Sécurité & Architecture

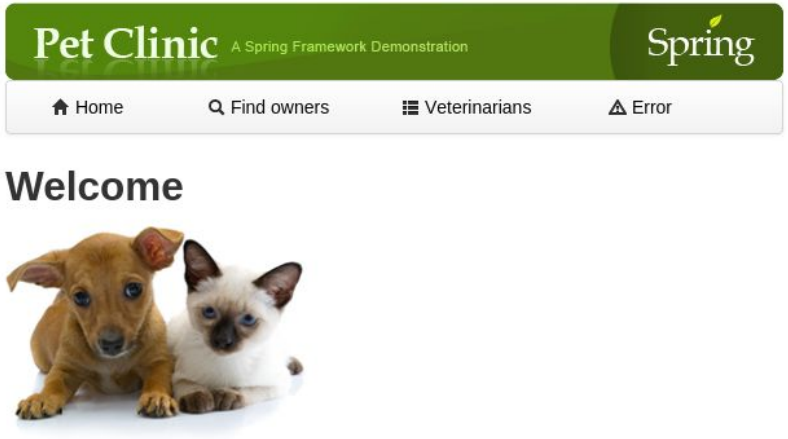
- Threat modeling & Risk-management
- Bonnes pratiques/Best practices
- Revues d’architecture
- Flou/Responsabilités
  - Validation des flux
  - Algorithmes de chiffrement?
  - Autorisation ⇒ Authentification
  - Non sécurisé pour faciliter

⇒ Impliquer votre responsable sécurité

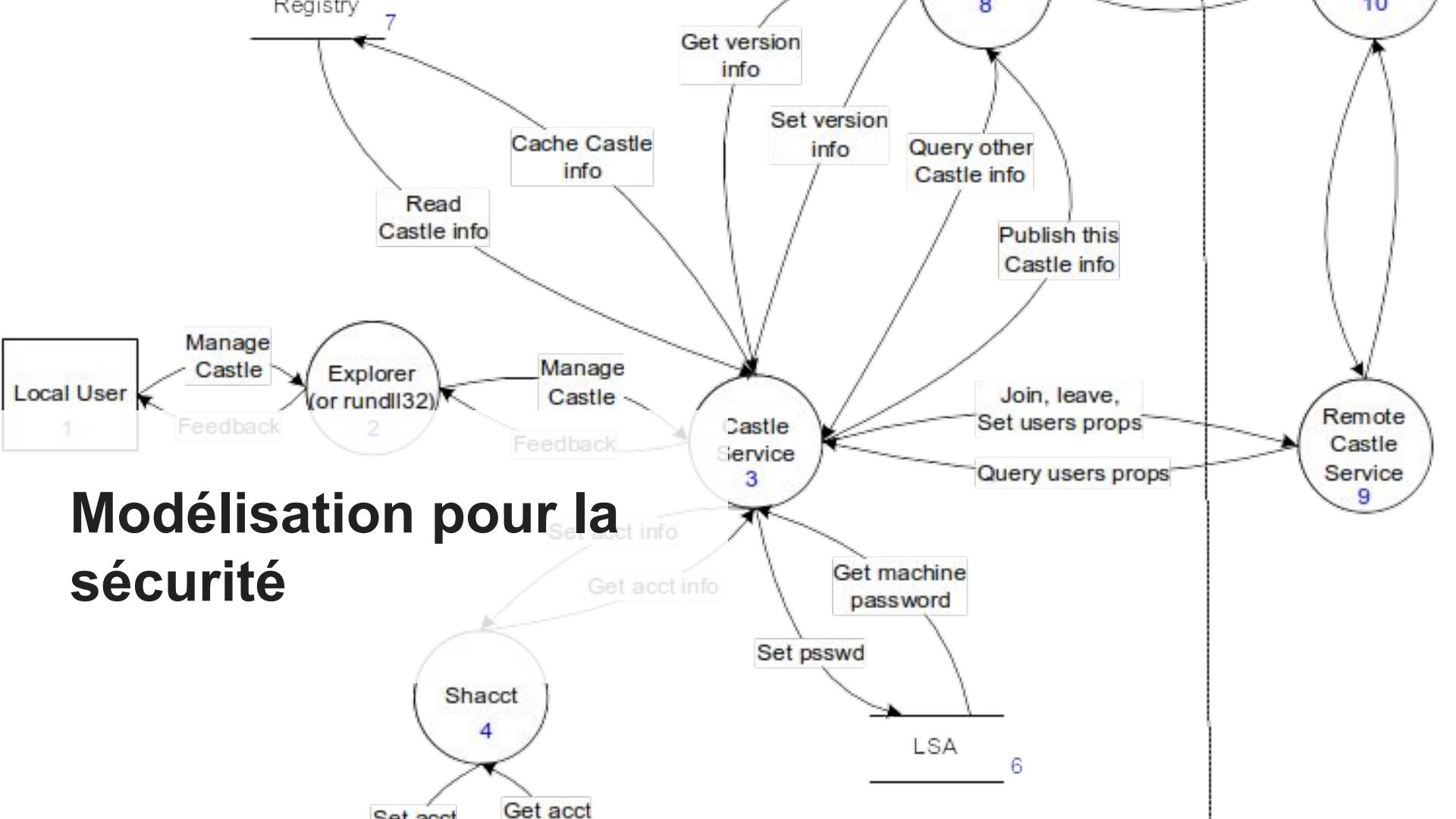
Pet Clinic

Revenons à notre clinique

- Prise de rendez-vous
- Carnet animal
- Paiement
- Formulaire de contact des vétérinaires



Des fonctions à surveiller?



Modélisation pour la sécurité

DFD

- Présenter les composants
- Les flux entre les composants
- Plusieurs niveaux de diagrammes
- Comprendre ce qui peut intéresser les attaquants
- Imaginer des scénarios d’attaques

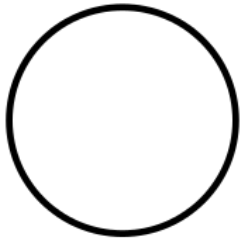
Surface d’attaque

Zone de confiance (Trusted boundaries)

Data Flow Diagram

Element	Apparence	Signification	Exemples
Processus	Rectangle arrondi, cercle concentrique, ...	Tout code exécuté	C, C#, Python, Java, ...
Flux de données	Flèche	Communication entre les processus ou avec la base	Echanges type HTTP, REST, SQL, ...
Stockage de données	Deux lignes parallèles avec un nom	Ce qui est stocké	Fichiers, BDD, LDAP, ...
Entité externe	Rectangle	People ou code en dehors de notre contrôle	L'utilisateur, partner.com, ...

Exemples



Function

File/Database

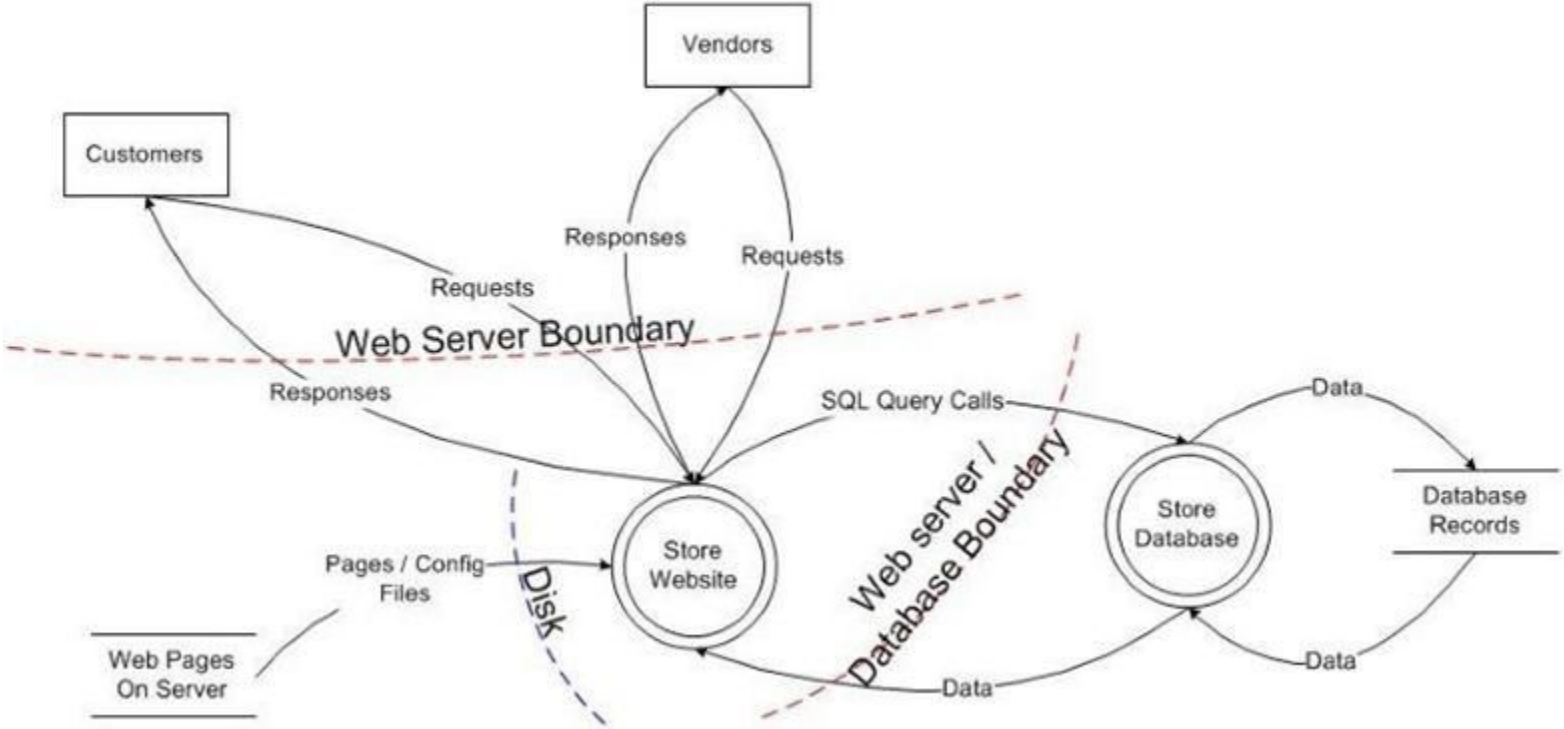
Input/Output

Flow

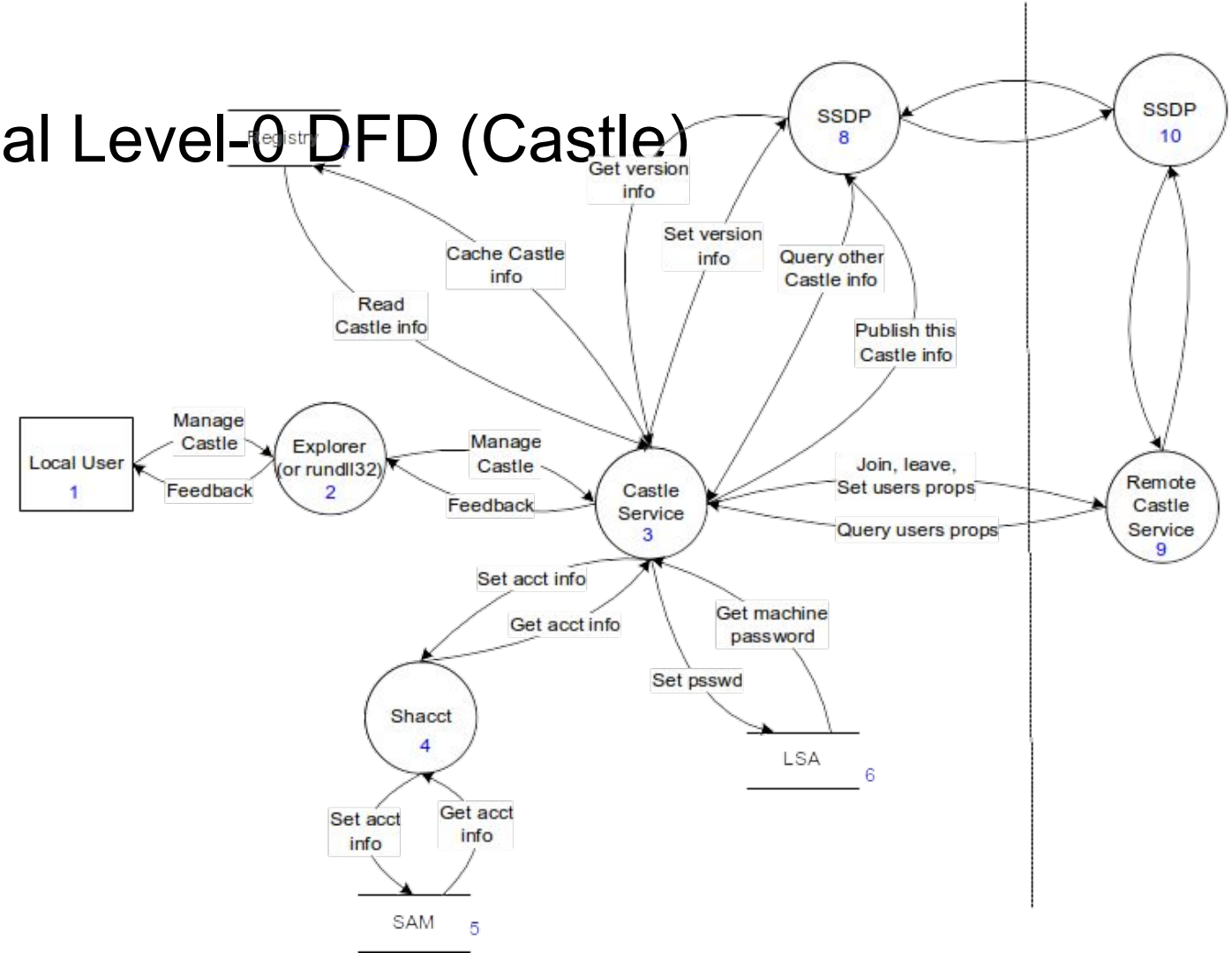
Trust Boundaries and Entry Points

- Frontières de confiance
- Différents niveaux de confiance
  - Avec internet : aucune
  - Inter-application : moyenne
  - Intra-application : élevé
- Menaces : autour de ces zones

Exemples


Vendors
Customers
Responses
Requests
Requests
Web Server Boundary
Responses
SQL Query Calls
Data
Web server / Database Boundary
Store Database
Database Records
Pages / Config Files
Disk
Store Website
Web Pages On Server
Data
Data

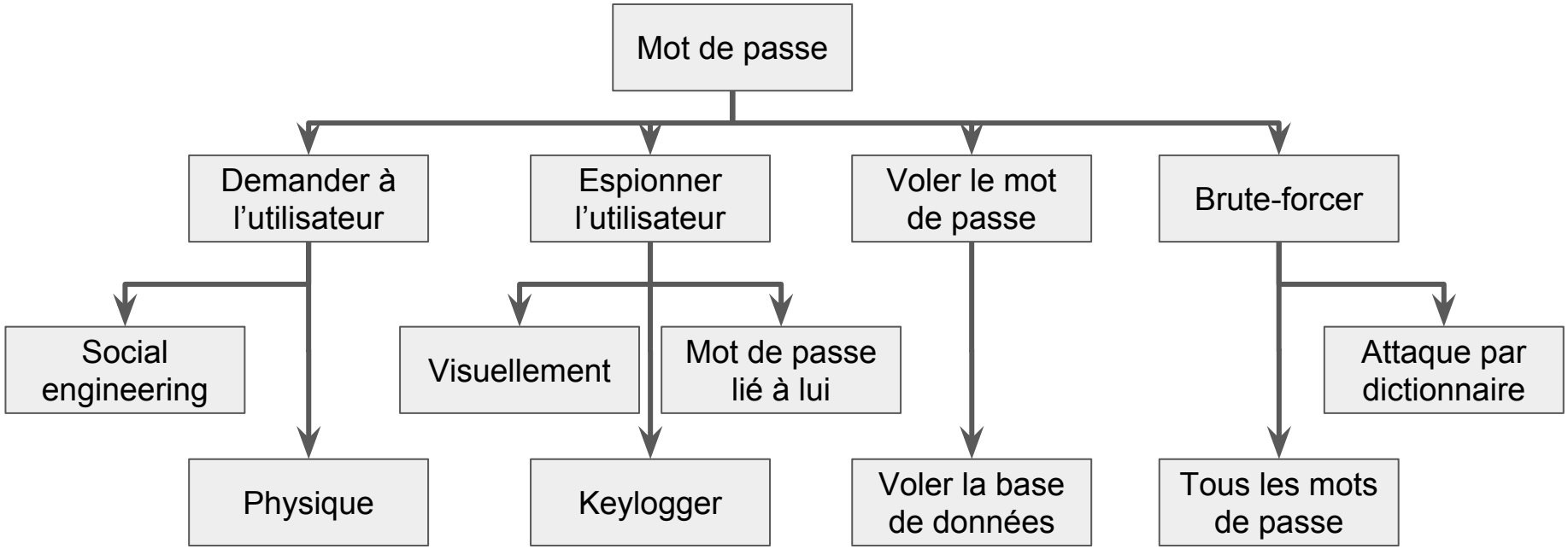
A Real Level-0 DFD (Castle)



Surface d’attaque et zones de confiance

- Zones de mélange entre données sûres et non sûres
  - Données sûres/non sûres (ex: WAN/LAN)
  - Changement dans le concept (ex: Argent “négatif”)
- Surface d’attaque =
  - Chemin
  - Code d’authentification, d’autorisation, …
  - Données utiles (secrets, données critiques, …)
  - Code les protégeant (Chiffrement, Auditing, …)
- **⇒ But: réduire la surface d’attaque**

Attack trees



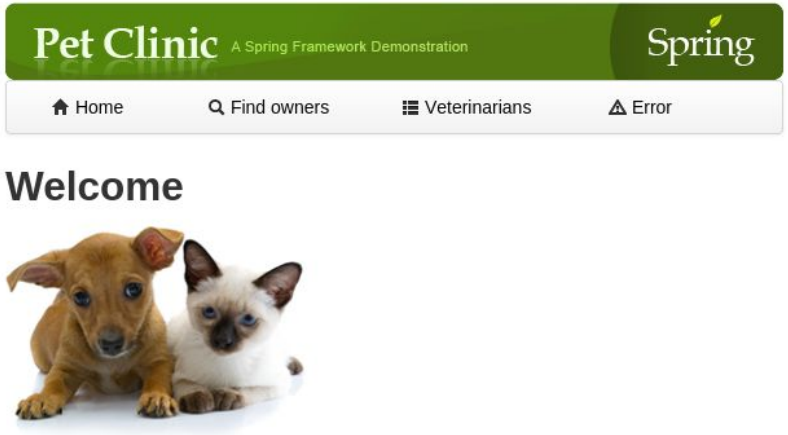
Exemple de résolution “Vol de la base de données”

- Empêcher le vol
  - Limiter les accès réseaux
  - Protéger contre les SQLi
  - Limiter les accès users
  - Déléguer à un service (Authorisation Server)
- Rendre inutile
  - Chiffrer la base
  - Chiffrer le mot de passe
  - Hasher et saler le mot de passe
  - Séparer mot de passe et sel en deux bases

Pet Clinic

Revenons à notre clinique

- DFD & Attack Surface
- Attack Tree



spring by Pivotal

Modèles de sécurité

Exemple de modèle de sécurité : JVM

- Java : langage de programmation (1990)
- Langage compilé en bytecode Java
- Bytecode exécuté dans une machine virtuelle : JVM

Exemple de modèle de sécurité : JVM

- Mémoire managée
- Vérification du bytecode
- Sandbox : exécuter du code non-sûr
  - Security Manager:Contrôle l’accès aux API, fichiers, …
- Librairie signée
- Java 9/Jigsaw : encapsulation
- Secure classloader
- JarSigner

Exemple de modèle de sécurité : UNIX

- Discretionary Access Control : User & Group
- ACL/File permission : rwxds
- Authentication : PAM
- Hardened Linux : SELinux & AppArmor
- OOM Killer
- Buffer Overflow Protection : GCC et -fstack-protector
- /etc/password shadowing

Réponses aux incidents

- Préparation
- Détection et analyse
- Containment
- Eradication
- Recovery
- Analyse Post Incident

Cloud et la sécurité

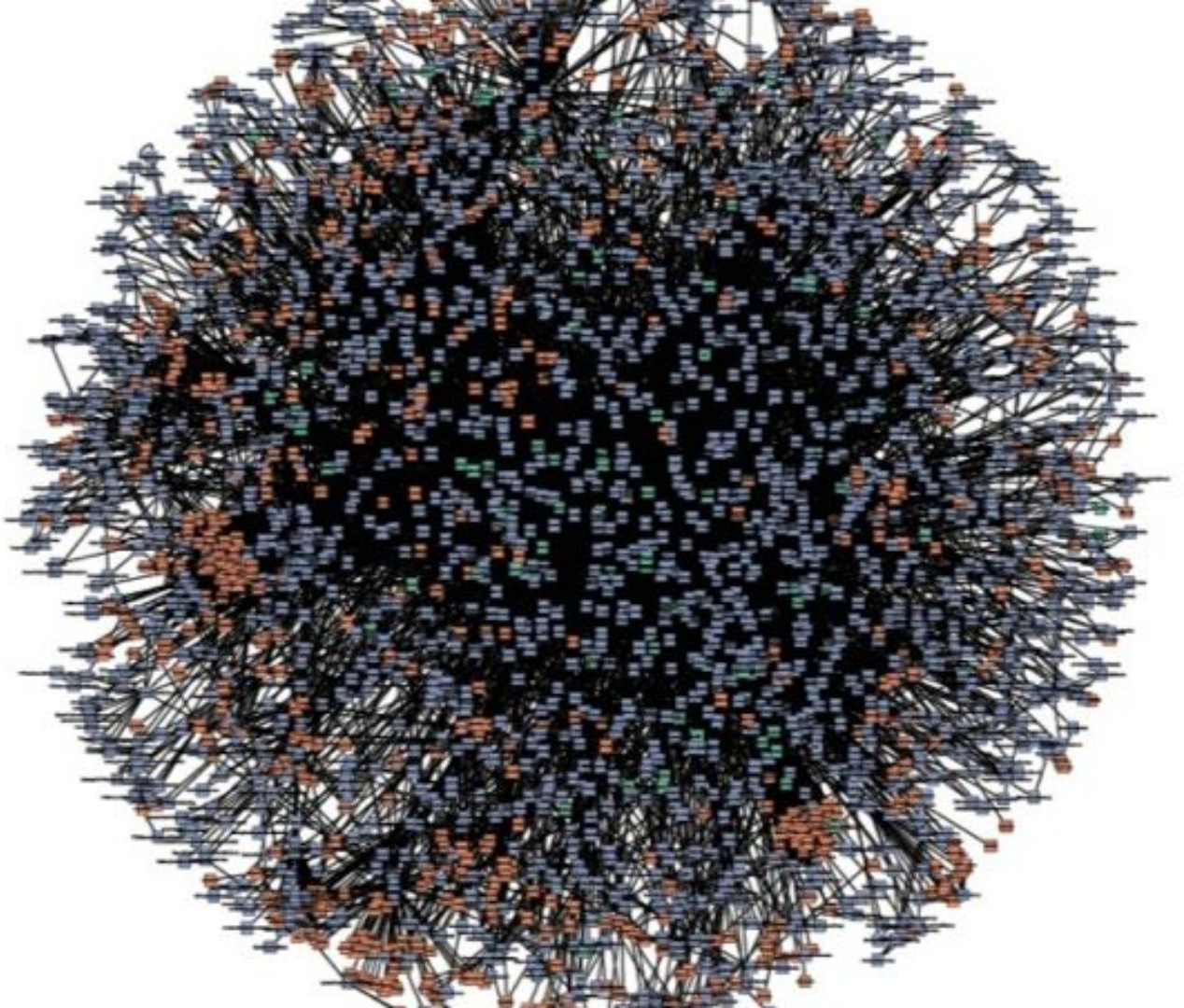
Cloud

- Gestion par un service tiers
- Nouvelles approches: Cloud Native, Microservices, ...
- Nouvelles fonctionnalités : scalabilité automatique, ...
- Nouvelles responsabilités :authentification,monitoring, ...
- Nouvelles problématiques: confiance, souveraineté, transparence, ...
- Nouvelles normes: Cloud Security Alliance

Différents modèles

- IaaS : Infrastructure as a Service
- PaaS : Platform as a Service
- FaaS : Function as a Service
- SaaS : Software as a Service

C’est quoi?



De nouvelles architectures : Microservices

- Authentification
- Autorisation
- Log
- Service Discovery
- Configuration
- Déploiement
- Cycle de vie

Microservices vs. Monolithe

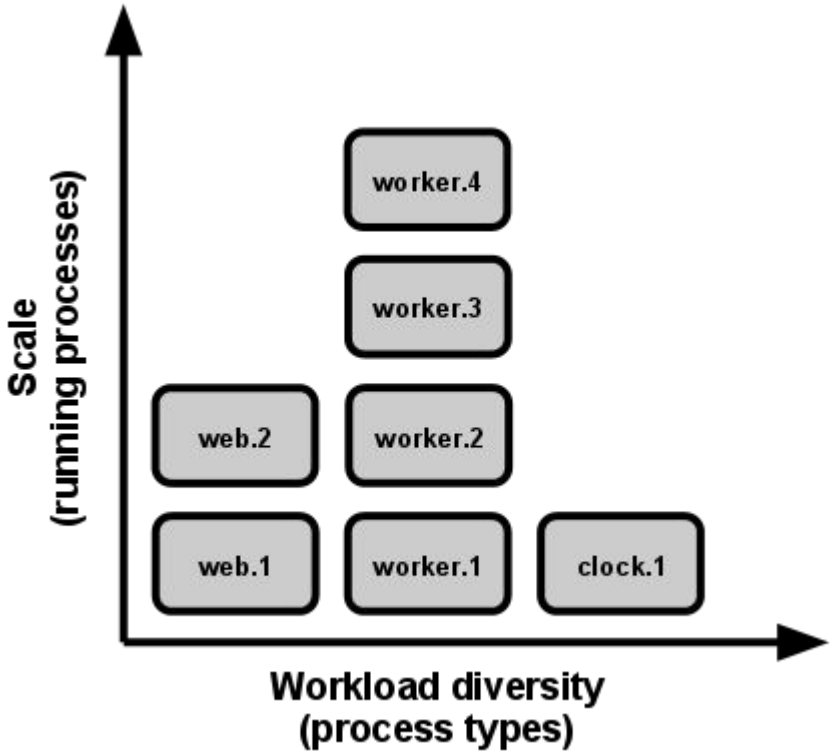
Monolithe

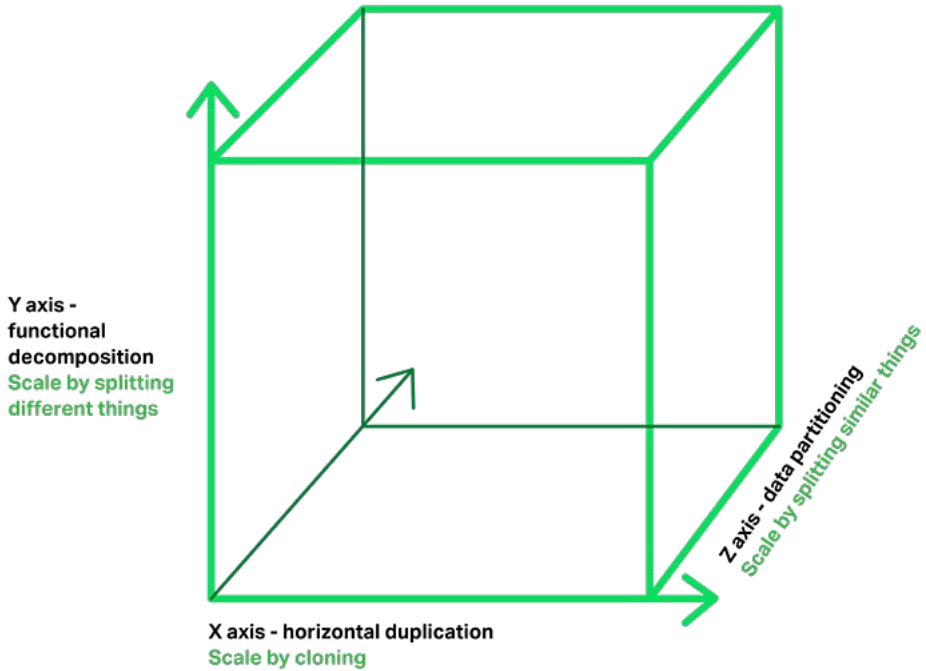
- Gros bloc
- Centré IHM
- Share everything
- Dependance entre composants
- MTBF
- 1 base
- Synchrone
- Scalabilité complexe
- Indisponible en cas de panne

Microservice

- Petit bloc
- Centré API
- Share nothing ⇒ duplicate
- Indépendance complète
- MTTR
- 1 base (SQL, NoSQL)/microservice
- Asynchrone ⇒ MQ
- Scalabilité facile
- Fonctionne même en cas de panne

Scalabilité





Microservices : nouveaux challenges?

- Economy of mechanism? Yes!
- Separation of duty? **Natif**!
- Auditability? **CorrelationId**
- Authentification? OAuth2
- FailSafe? Obligatoire
- Disponibilté? Scalable!
- Gestion des secrets?
  - Pas de mots de passe “en dur”
  - Configuration automatique
  - Vault
- Testable dans sa globalité? Non
- Concepts “partagés”? Client vs. Principal
- Intégrité?
  - Eventual Consistency
  - Share nothing ⇒ Duplication?

Cloud : Enabler ou Weakness

- Offre des solutions de sécurité intégré
  - Authentication et Rôles (AWS Cognito), Protection DDoS, Certificats, Serveurs à jour, Vérifications de la configuration...
  - DDoS ⇒ Scalable, ...
- Nouvelles sources de failles
  - Maîtrise par les équipes
  - Cloud sécurisé?

Containment, Eradication, Recovery

- Containment : empêcher la menace de s’étendre
- Eradication : détruire la menace
- Recovery : “soigner” les plaies

Rotate, Repave, Repair

- Rotate - secrets every few minutes
- Repave - every server and application in the datacenter every few hours
- Repair - vulnerable operating systems and application stacks consistently within hours of patch availability