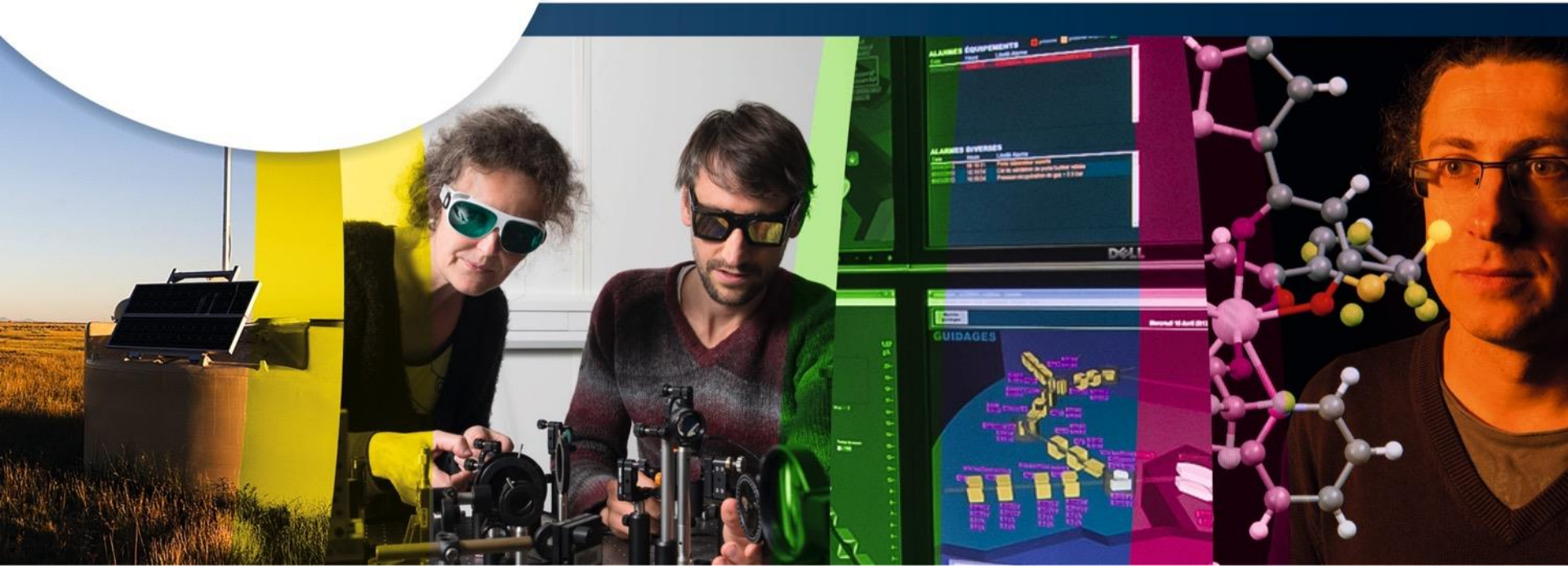
www.cnrs.fr

# Principes



# Principes

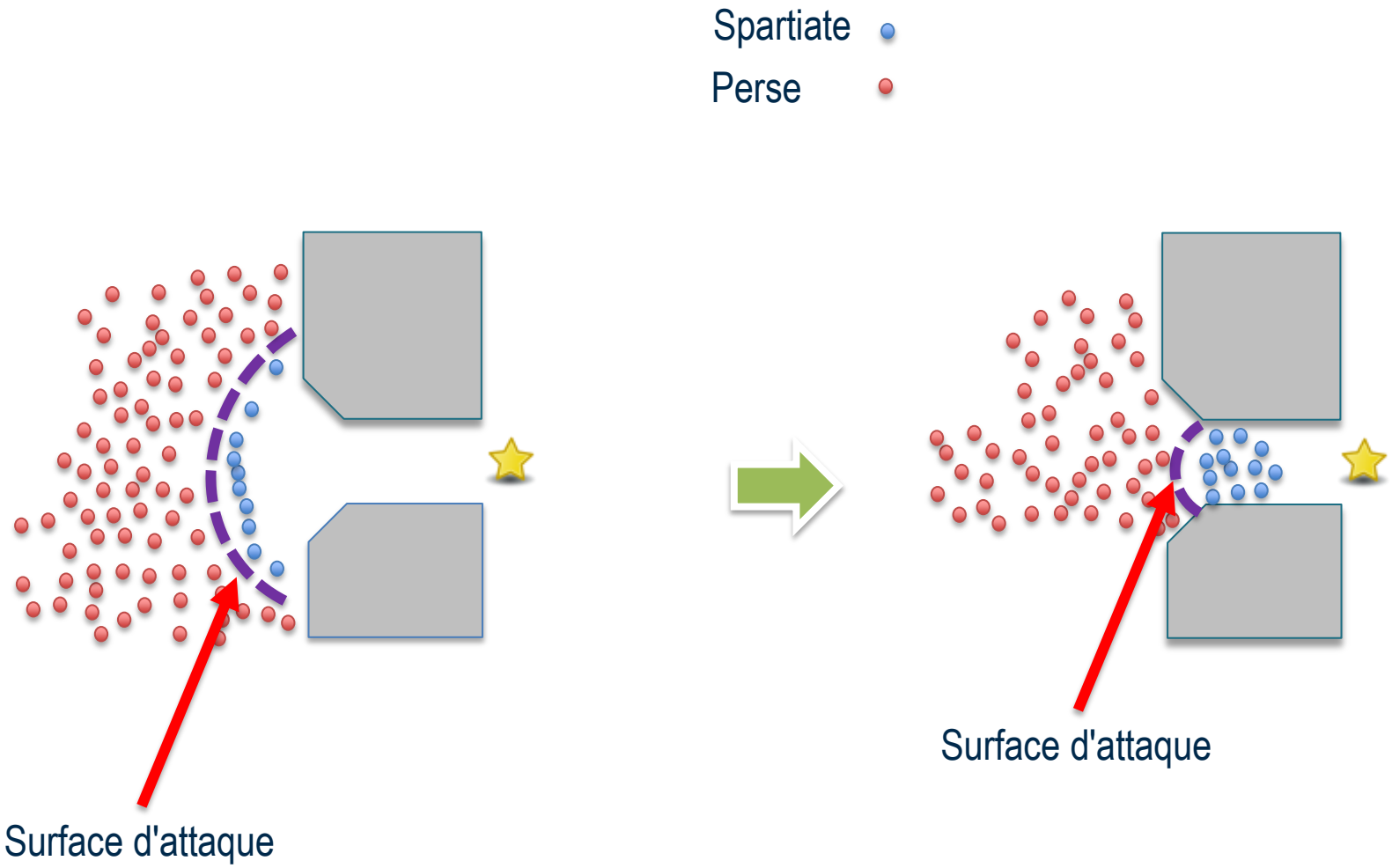
www.cnrs.fr

- **Qu'est-ce qu'un principe ?**
  - Une orientation résumée en une phrase
  - Une boussole en cas de doute ou de désorientation
  - Le monde de l'information est féru de principes
    - Loi de Demeter (principe de moindre connaissance)
    - Don't Repeat Yourself
    - Worse Is Better
- Les principes sont des préceptes qui servent
  - à façonner un esprit critique,
  - à avoir une vision,
  - à se questionner.

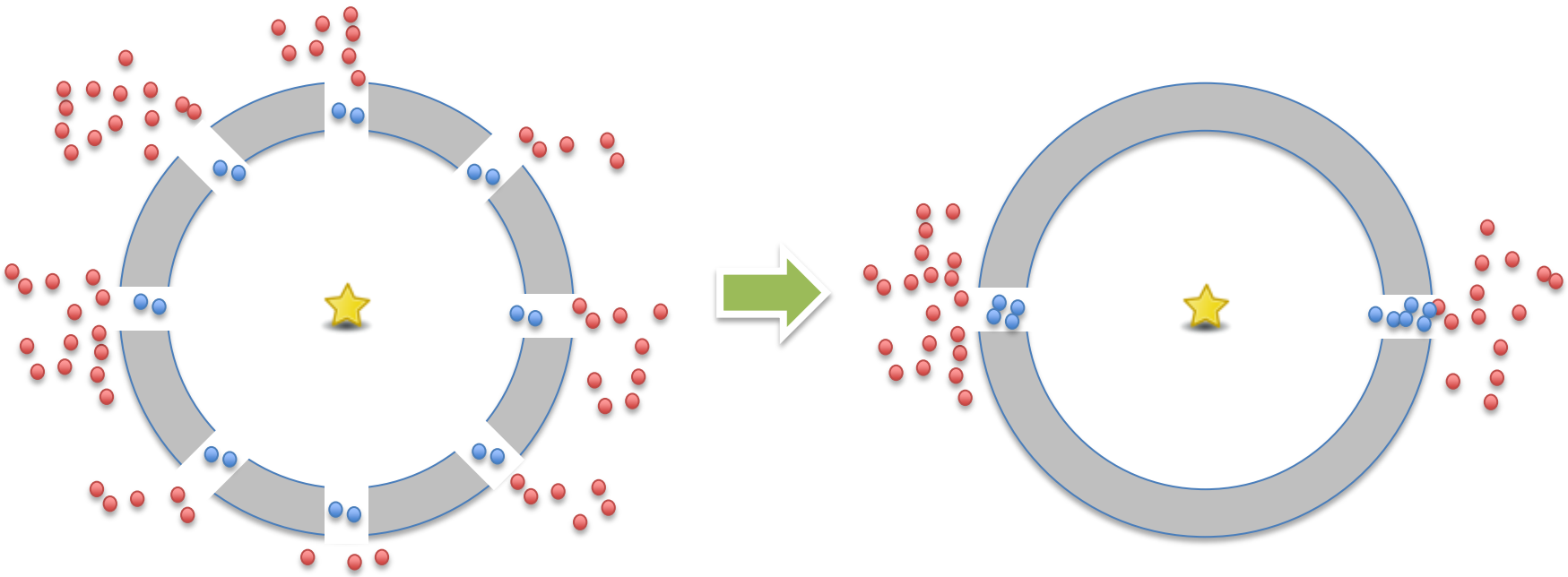
# Réduction de la surface d'attaque
## Attack Surface Reduction

- *Surface d'attaque*
  - Tous les moyens extérieurs et permettant d'agir de façon directe ou indirecte sur les fondamentaux de l'application
- **Signification du principe**
  - Il est plus facile de défendre un nombre réduit de points d'accès
    - de contrôler une petite surface qu'une grande aire
    - de défendre l'accès à une ville via un corridor qu'une plaine
- **Exemples**
  - Services réseaux inutiles
  - Fonctionnalités applicatives superflues
  - Rôles utilisateur sans pertinence
  - Web Service tiers à faible valeur ajoutée

# Les 300 spartiates à la bataille des Thermopyles



# Le nombre de points d'accès à un stade



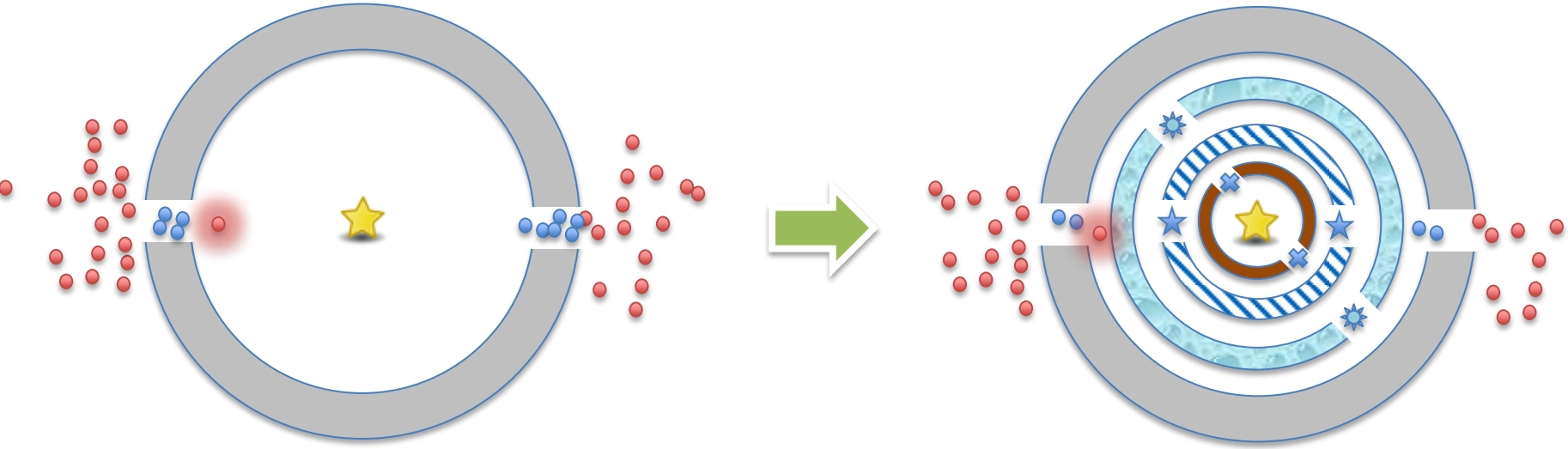
# Réduction de la surface d'attaque

- **Comment**
  - Enumérer l'essentiel, supprimer le superflu.
  - Lutter contre "Qui peut le plus peut le moins"
  - YAGNI: *"you ain't gonna need it"*
- **Références**
  - OWASP Attack Surface Analysis Cheat Sheet
    - Focalisation et évaluation par famille (bucket) pondérée

# Défendre en profondeur
## *Defend In Depth* ou *Ceintures/bretelles*

- **Signification du principe**
  - Gérer le risque avec des lignes de défense successives de façon à ce que, si une ligne cède, la suivante puisse parer à l'attaque et ainsi de suite.
- **Exemples**
  - Un sas, puis une porte blindée, puis un coffre à code
  - Enceintes de confinement d'un réacteur nucléaire
  - Un Firewall réseau, un firewall au niveau du serveur, un Web Application Firewall
- **Contre-exemple**
  - La maison en paille du 1er petit-cochon
  - Se croire protégé sur le réseau local du labo et laisser mysql avec "root", ""
  - Croire qu'un utilisateur authentifié est légitime pour faire une action sans vérifier

# Défendre en profondeur

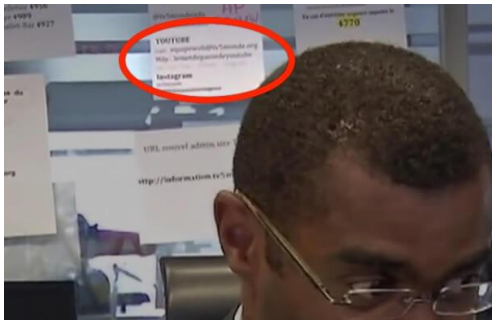


# Défendre en profondeur

- **Comment**
  - Aligner les solutions de défense de même nature
- **Attention**
  - **Coût raisonnable** : problème de la débauche de moyen ➔analyse de risque, modélisation des menaces pour choisir la stratégie la plus adaptée
  - **Utilisabilité** : si la stratégie rend le système à peine utilisable, les utilisateurs s'en détourneront ou dégraderont la sécurité, la finalité métier ne sera plus remplie
- **Références**
  - https://www.us-cert.gov/bsi/articles/knowledge/principles/defense-in-depth
  - l'ANSSI propose un guide très complet sur la défense en profondeur

# Secure the Weakest Link

- **Signification du principe**
  - La sécurité possède la résistance de son maillon le plus faible
  - L'attaquant concentrera ses efforts pour l'élément le plus faillible
- **Exemples**
  - Une application très bien conçue communique avec une base de données distante en clair
  - Un détenu derrière plusieurs murs d'enceinte, cour de promenade à ciel ouvert ➔Évasion par hélicoptère
  - L'application est blindée, le niveau de sécurité élevé… mais le mot de passe a été filmé et diffusé à la TV …



- **Cohérence avec la défense en profondeur**
  - *La défense en profondeur s'applique à des défenses* ***de même nature***
  - *Le principe de sécurisation du lien faible s'applique à des défenses* ***de nature différente****.*

# Secure the Weakest Link

- **Comment**
  - Veiller à avoir une politique de sécurité cohérente et homogène
  - Ne pas se focaliser sur des détails et avoir en tête l'image globale
  - Faire des revues de sécurité
  - Le maillon faible est souvent humain…

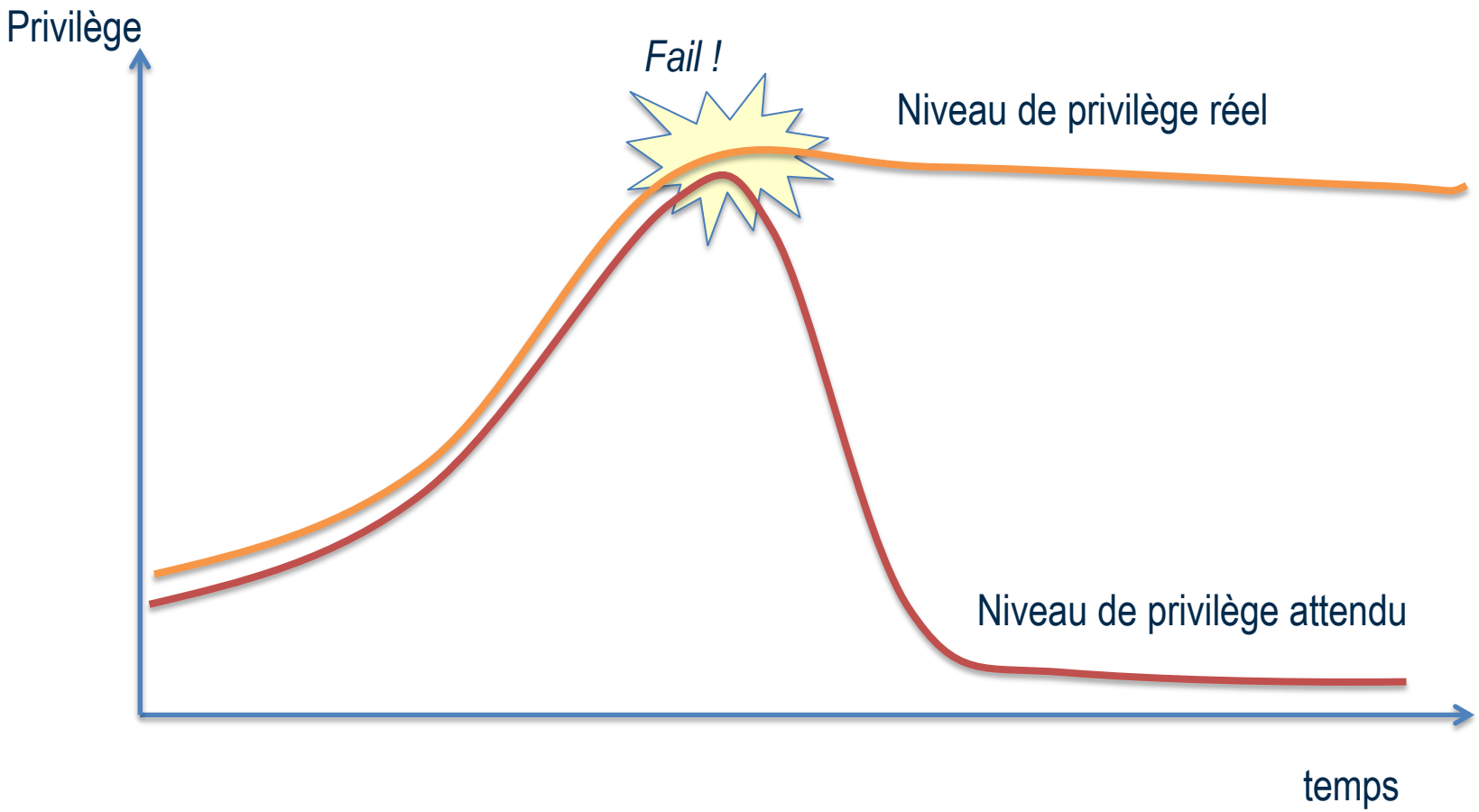
- Références
  - https://www.us-cert.gov/bsi/articles/knowledge/principles/securing-the-weakest-link

# Fail Securely

- **Signification du principe**
  - Toute défaillance du système doit aboutir à une situation sécurisée
  - Aucune défaillance ne doit aboutir à une élévation de privilège

- **Illustration**
  - Laisser les clefs sur la porte en rentrant pour répondre  à un coup de fil
  - Laisser un utilisateur avec un droit admin après l'explosion de la connexion à la base de données

# Fail Securely



# Fail Securly
# avec des exceptions

```
function boolean isAdminForResource(User user, Action action,
                                    Resource resource, Data data) {
    boolean isAdmin = false;

    try {
        isAdmin = isPermissionOk(user, action, resource, data);
    } catch(InvalidAuthenticationException ipe) {
        doSomething();
    }
    return isAdmin;
}
```

# Fail Securly
# avec des codes retours

```
/**
 * @return ACCESS_DENIED, DB_ERROR, OUT_OF_MEMORY, IS_ACCESS_OK
 */
function isAccessOk(...) {
 (...)
 else {
    return IS_ACCESS_OK;
 }
}

$isOk = isAccessOk(...);
if ( $isOk === ACCESS_DENIED ) { // et si $isOk === DB_ERROR ?
    // FAIL!
    informUser();
} else {
    performTask();
```

# Fail Securly
# avec des codes retours

```
}
/**
 * @return ACCESS_DENIED, DB_ERROR, OUT_OF_MEMORY, IS_ACCESS_OK
 */
function isAccessOk(...) {
 (...)
}

$isOk = isAccessOk(...);
if ( $isOk === IS_ACCESS_OK ) {
    performTask();
} else {
    informUser();
}
```

# Fail Securely

- **Comment faire ?**
  - Nier le privilège puis ne l'attribuer qu'après la vérification de toutes les conditions nécessaires
  - Utiliser une sémantique précise et explicite
- **Oui mais …**
  - Politique de prix chez Amazon (le 1er prix affiché est le bon même si il est faux),
- **La réponse est dans l'analyse de risque**, et non dans des positions basées sur nos valeurs personnelles
  - Risque1: "Perdre de l'argent sur une vente"
  - Risque 2: "Le client attend trop et se détourne vers un autre site de vente"

- **Références**
  - https://www.us-cert.gov/bsi/articles/knowledge/principles/failing-securely

# Principe de moindre privilège

- **Signification du principe**
  - Tout composant ou utilisateur doit opérer avec le minimum de permissions nécessaires à l'exécution de sa tâche.
  - Eviter que la compromission d'un processus bénéficie du niveau de privilège associé
  - Eviter les interactions avec des instances de même niveau de privilège en cas d'usurpation
- **Exemple**
  - Le principe militaire *"need to know"*
- **Contre-Exemple**
  - Faire tourner le serveur web en *root*
  - `chmod -R 777 *`
  - Donner les droits de modification à un utilisateur en lecture seule

# Principe de moindre privilège

- **Comment**
  - Penser en terme de stricte nécessité
  - Faire une matrice Rôle/Privilège exhaustive
  - Différencier les fonctions en lecture et écriture
    - *Pour appliquer un privilège de lecture seule, appeler une **fonction** en lecture seule.*
    - Utiliser le pattern CommandAndQuery
  - N'accorder le privilège nécessaire que le temps de l'action, **puis le retirer de suite**
- **Attention**
  - Ne pas aller à l'encontre des besoins métier et les rendre absurdement difficiles.
- Penser au principe **Responsabilité/Surveillance** (Octo)
  - La sur-sécurité lèse l'utilisateur et ne bloque pas l'attaquant
  - "Tu es responsable et par défaut nous te faisons confiance"
  - "Mais nous surveillons toutes tes actions et tu devras en répondre"

# Keep It Simple and Stupid

- **Signification du principe**
  - La complexité augmente le risque. Rester simple permet de contenir le risque
  - Est sécurisable ce qui est appréhendable

- **Contradiction avec le principe de Défense en Profondeur**
  - Trouver l'équilibre entre la superposition de couches de sécurité, mais ne pas rendre complexe la couche en soi (*Viega and McGraw*)

- **Comment**
  - Etablir une cartographie de l'application
    - Simplifier les "quartiers" trop complexes en ensembles plus simples
  - Eviter des fonctionnalités "cachées" dont la simple connaissance permet de contourner ces accès
    - Cela se sait tôt ou tard et ce sont autant de **backdoor** .

# Démarche de simplification (McGraw)

- **Dispersion** de mécanismes de sécurité dont l'appréhension devient difficile
- ➔limiter à des points de passage
  - délimités, simples, bien compris et maîtrisés
  - à travers lesquels le contrôle s'effectue.
- Cette démarche permet de mieux contrôler, vérifier et logger

- **Oh Wait….  KISS ➔Réduction de la surface d'attaque**

# Be reluctant to trust

- **Signification du principe**
  - N'accordez pas facilement votre confiance.
  - La confiance est transitive.
    - Si A a confiance en B et si B a confiance en C, alors A a confiance en C
  - ***Don't trust anyone***
- **Comment**
  - Vérifier vos dépendances, vos entrées/sorties, l'identité de votre base de données, votre DNS, …
  - Vérifier votre propre code !
    - Faites des tests unitaires, faites des revues

# Open Design

- **Signification du principe**
  - La sécurité d'un système ne doit pas reposer sur l'obscurité de sa conception, mais sur un secret.
  - Si les détails de conception sont connus, c'est alors une catastrophe
- **Contre-Exemple**
  - Mettre des clefs en dur et compter sur la compilation ➔décompilateur
  - La sécurité reposant sur l'obfuscation est toujours contournée par rétro ingénierie
- **Comment**
  - Concevez votre application comme si son code était accessible à tous.

# Make security usable

- **Signification du principe**
  - La sécurité doit être utilisable. Si elle est trop complexe ou trop pénalisante, elle sera contournée.
  - *Make sure that your security system is as secure as it needs to be, but no more. If you affect usability too deeply, nobody will use your stuff, no matter how secure it is. Then it will be very secure, and very near useless.*
- **Exemple**
  - Une politique de mot de passe très difficile avec des changements fréquents ➔l'utilisateur finira par l'écrire sur un post-it
  - Un accès très pénible pour accéder à l'agenda du service
- **Comment**
  - "Designers are not the users"
  - La sécurité n'est (presque) jamais la priorité de l'utilisateur

# Toujours revérifier
## Mediate completely

- **Signification du principe**
  - Toute autorisation doit être revérifiée à chaque accès à une ressource
  - Une autorisation ne doit jamais être considérée comme toujours acquise
- **Exemple**
  - Accéder à un coffre-fort nécessite une vérification même si vous êtes accompagné du directeur de la banque, car il peut avoir été trompé
  - L'exploitation d'une faille sur un contrôleur permet au niveau d'un service de modifier la base de données
- **Comment**
  - Spécifier et vérifier les autorisations à chaque opération de modification
  - Utiliser des annotations
  - ***Optimization is evil***
    - *L'overhead est négligeable*
    - *Faire attention à la mise sous cache*

# Exemple avec Symfony

```
function isAccessOk(...) {

use JMS\SecurityExtraBundle\Annotation\Secure;
// ...

class NewsletterManager
{

    /**
     * @Secure(roles="ROLE_NEWSLETTER_ADMIN")
     */
    public function sendNewsletter()
    {
        // ...
    }
```

# Exemple avec Spring Security

```
/**
 * Returns a single traveller based on its uid
 *
 * @param uid              the uid of the desired traveller
 * @param onlyActiveRoles indicate whether only active roles should be returned
 * @return this traveller resource (404 if it does not exist)
 */
@RequestMapping(method = GET, value = "/travellers/{uid}")
@PreAuthorize("#uid == authentication.name or hasPermission(#uid,'adminOrManagerOnPerimeter')")
public ResponseEntity<Traveller> getTraveller(
                    @PathVariable String uid,
                     @RequestParam(defaultValue = "false") boolean onlyActiveRoles) {
     ...
     }
```

# Les autres

- Separation of privilege
  - Système de déclenchement du feu nucléaire avec plusieurs clefs partielles
  - https://www.us-cert.gov/bsi/articles/knowledge/principles/separation-of-privilege
- Segregation of duty
  - Un unique individu ne peut ordonner une action et en bénéficier
  - https://en.wikipedia.org/wiki/Separation_of_duties
- …

- https://cryptosmith.com/2013/10/19/security-design-principles/
- http://searchsecurity.techtarget.com/opinion/Thirteen-principles-to-ensure-enterprise-system-security

# Fin

www.cnrs.fr