Le système d'exploitation CLIP

Genèse, principes et limites

Vincent Strubel

ANSSI / SDE

1^{er} juin 2017

Qu'est-ce que CLIP ?

Projet de recherche ANSSI sur la sécurisation d'un OS

- Évolution graduelle : maquette, prototype puis "produit"

Un système d'exploitation durci sur base Linux

- Forte résistance aux codes malveillants
- Sécurité dans la durée : mitigation des attaques et mises à jour

Qu'est-ce encore que CLIP ?

Un système d'exploitation multiniveau

- Cloisonnement de deux niveaux de sensibilité

Une "méta-distribution"

- Distribution Linux complète
- Sources communes, plusieurs distributions binaires

Historique

- 2005 : maquettage initial (sur base FreeBSD)
- 2006-2008 : prototype opérationnel
  - Portage sur une base Linux
- 2009 : certification EAL2+, qualification standard
- Depuis 2009 : déploiement et amélioration fonctionnelle
  - Premiers déploiements dans l'administration
  - Mise à disposition de partenaires industriels
- 2015 : premiers POC de déploiements chez des OIV du privé

Principes de sécurité

Principes de sécurité
Utilisation du cloisonnement

Principe du cloisonnement

Le cloisonnement par *containers* est au coeur de CLIP

- Virtualisation légère : un noyau, plusieurs couches utilisateur (*userland*)
- Isolation de groupes de processus et de leurs ressources

Découpage en compartiments étanches : "cages"

- But : limiter la portée d'une attaque réussie

Rappel : *containers* Linux

Partitionnement par *namespaces*

- Niveau d'indirection supplémentaire, partitionnement des ressources en plusieurs espaces isolés
- *chroot* étendu à toutes les ressources

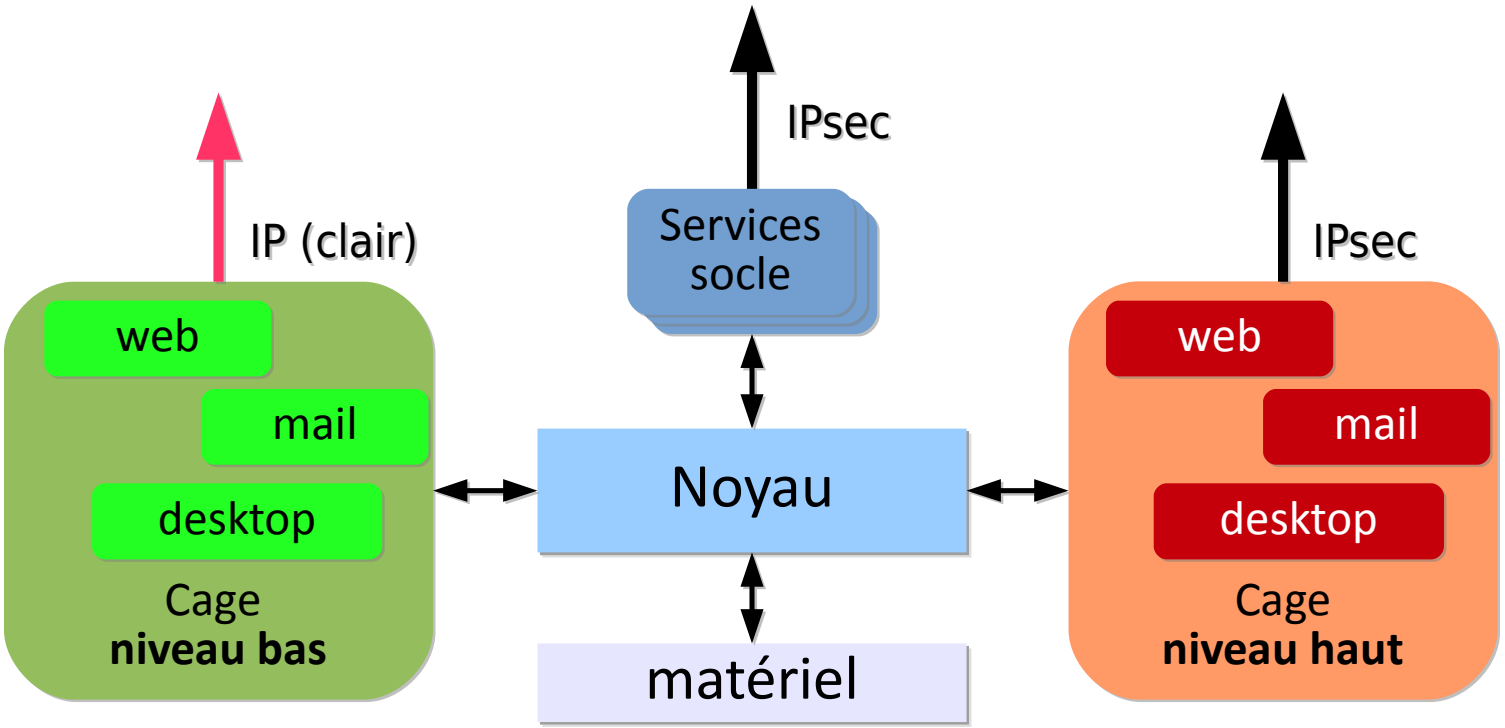
Réduction des privilèges via les capacités Linux

- Découpage des privilèges *root* en plusieurs capacités
- Capacités masquées dans les *containers* même pour *root*

Cloisonnement multiniveau

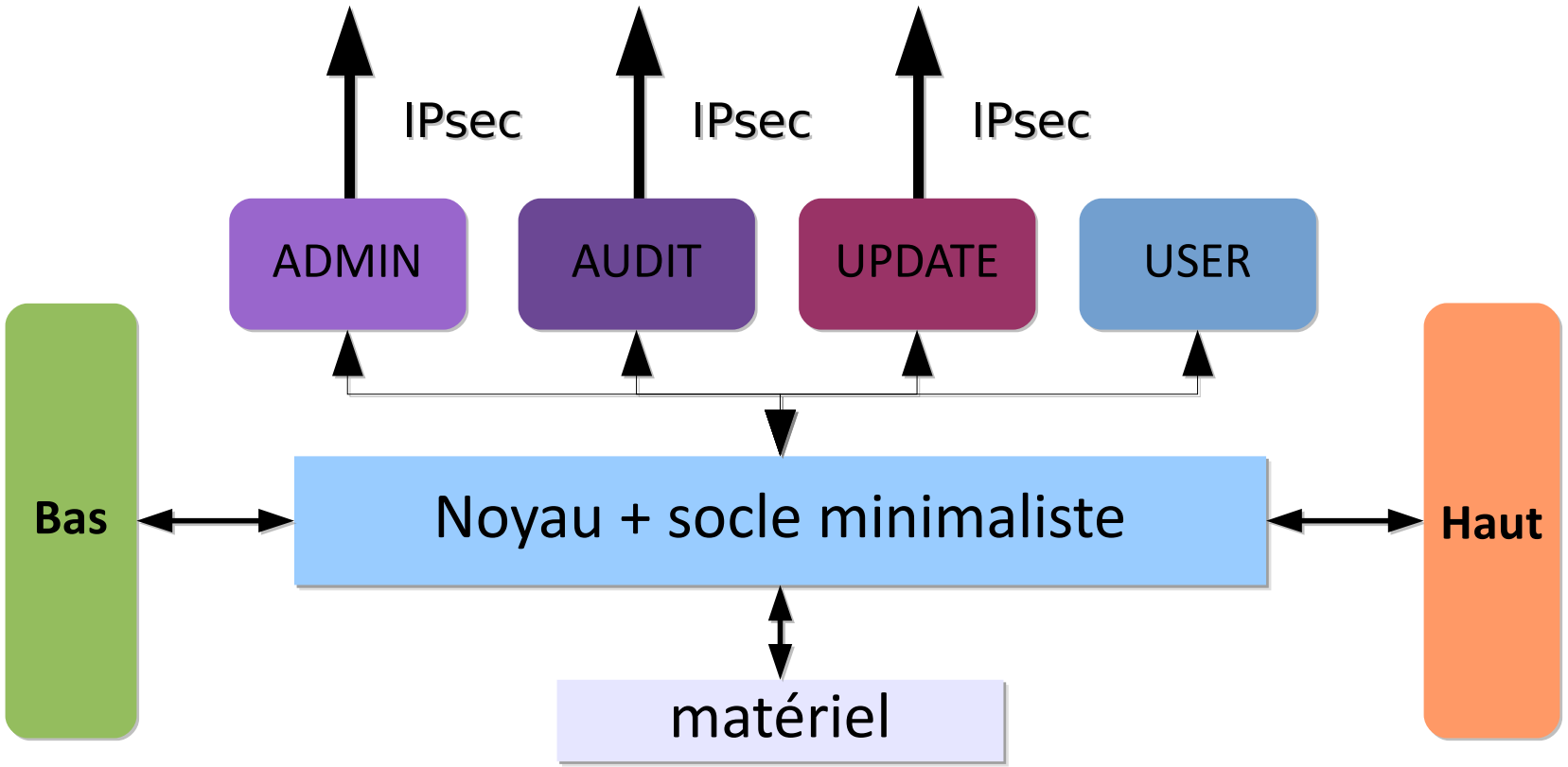
Deux environnements logiciels complets isolés entre eux

- Niveau bas connecté directement au réseau
- Niveau haut avec un VPN obligatoire



Cloisonnement par rôles

Séparation stricte des rôles : un compartiment dédié par rôle



Cloisonnement des services

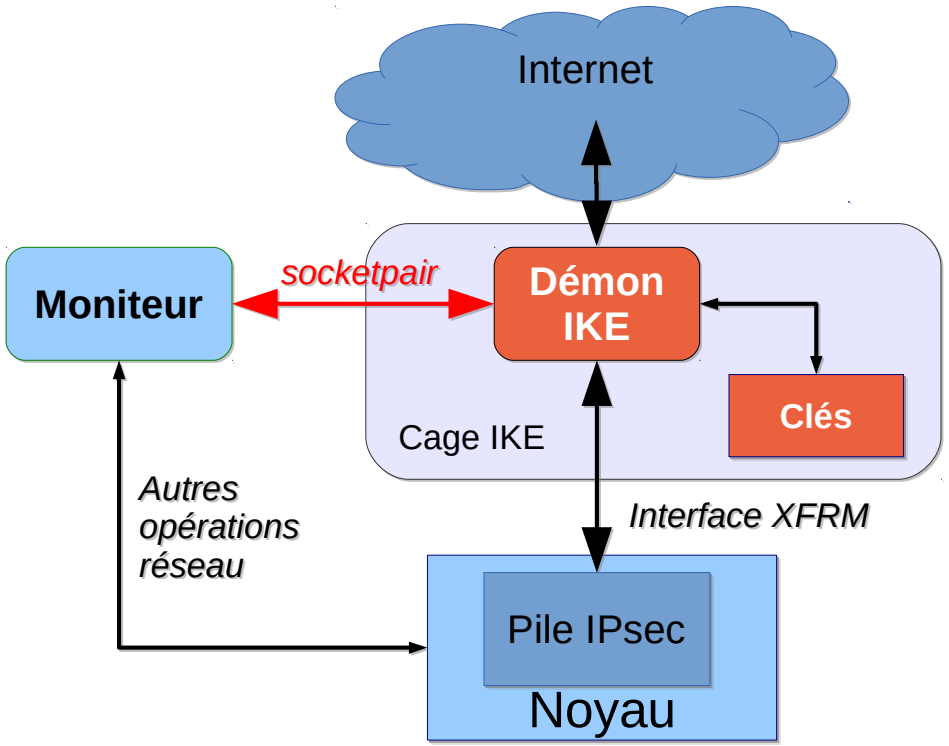
Réduction des privilèges des services critiques

- Cage dédiée pour chaque service, avec auto-enfermement
- Comme un *chroot* en mieux :
  - Cloisonnement étendu à toutes les ressources
  - Accès limité au réseau
  - Réduction imposée des capacités

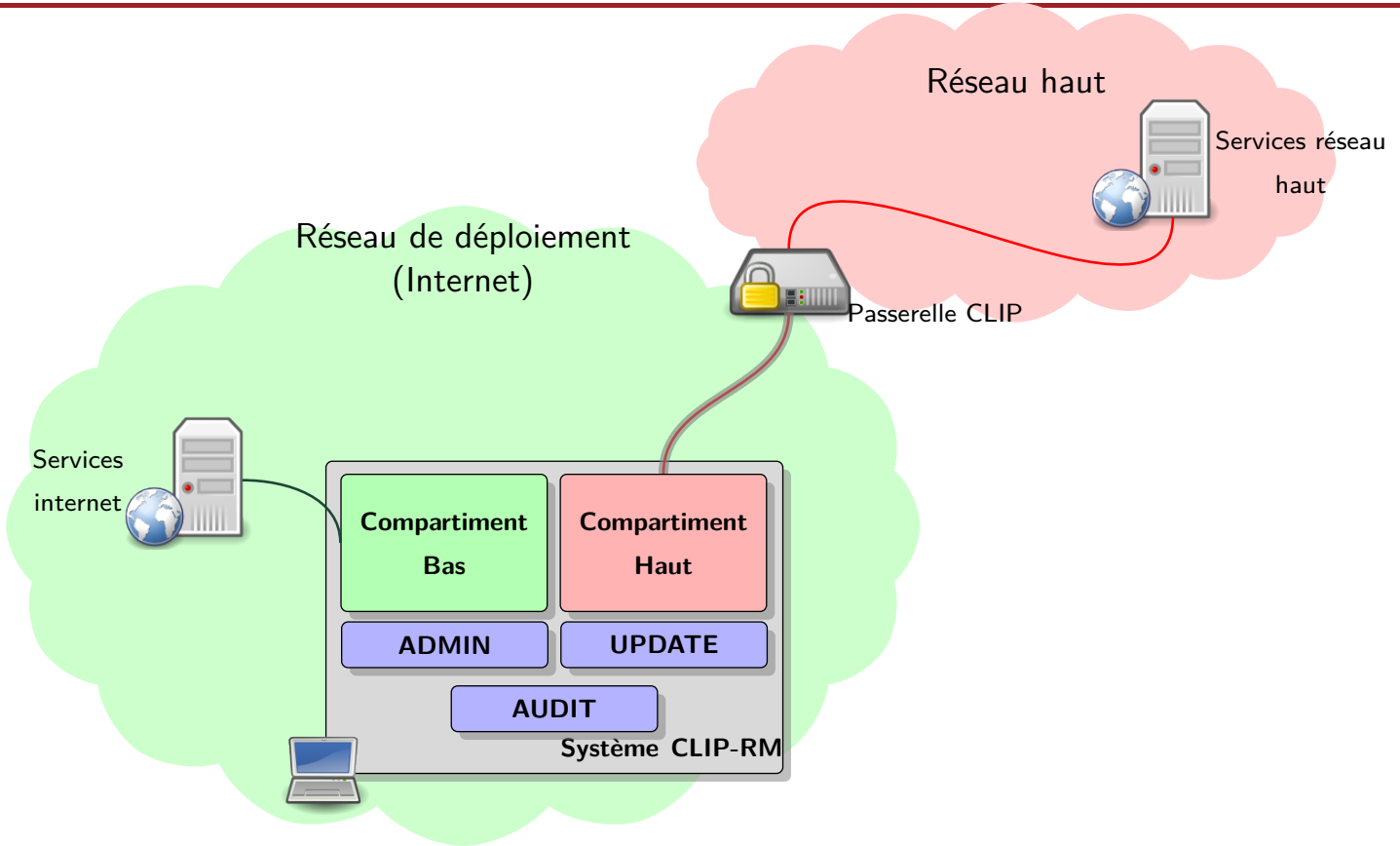
Effort justifié pour les services exposés ou privilégiés,
par ex. services réseau ou serveur graphique

Exemple : IKE

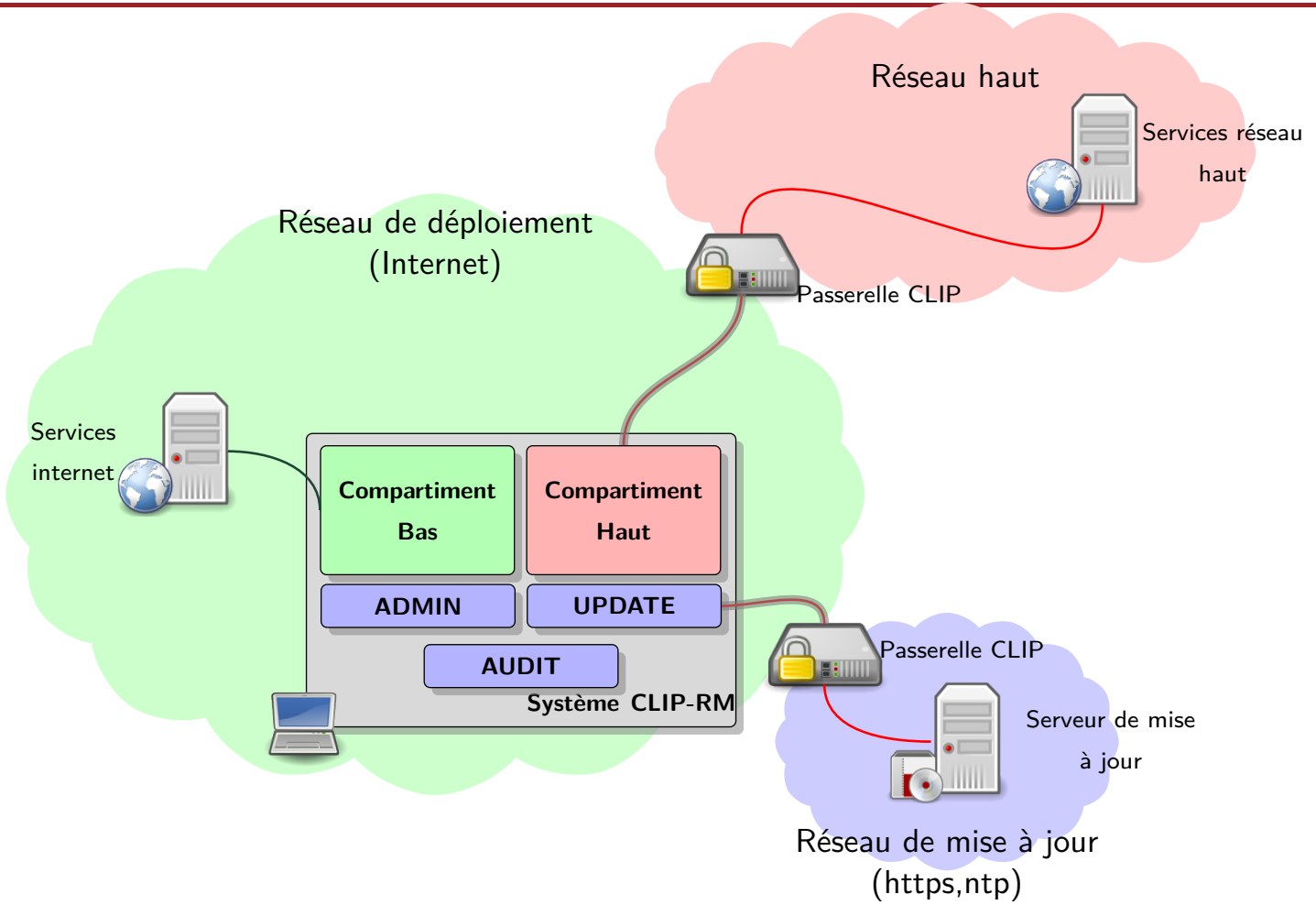
- Démon particulièrement privilégié et exposé
- Réduction de privilèges par "encagement"
  - On pourrait sans doute faire mieux...



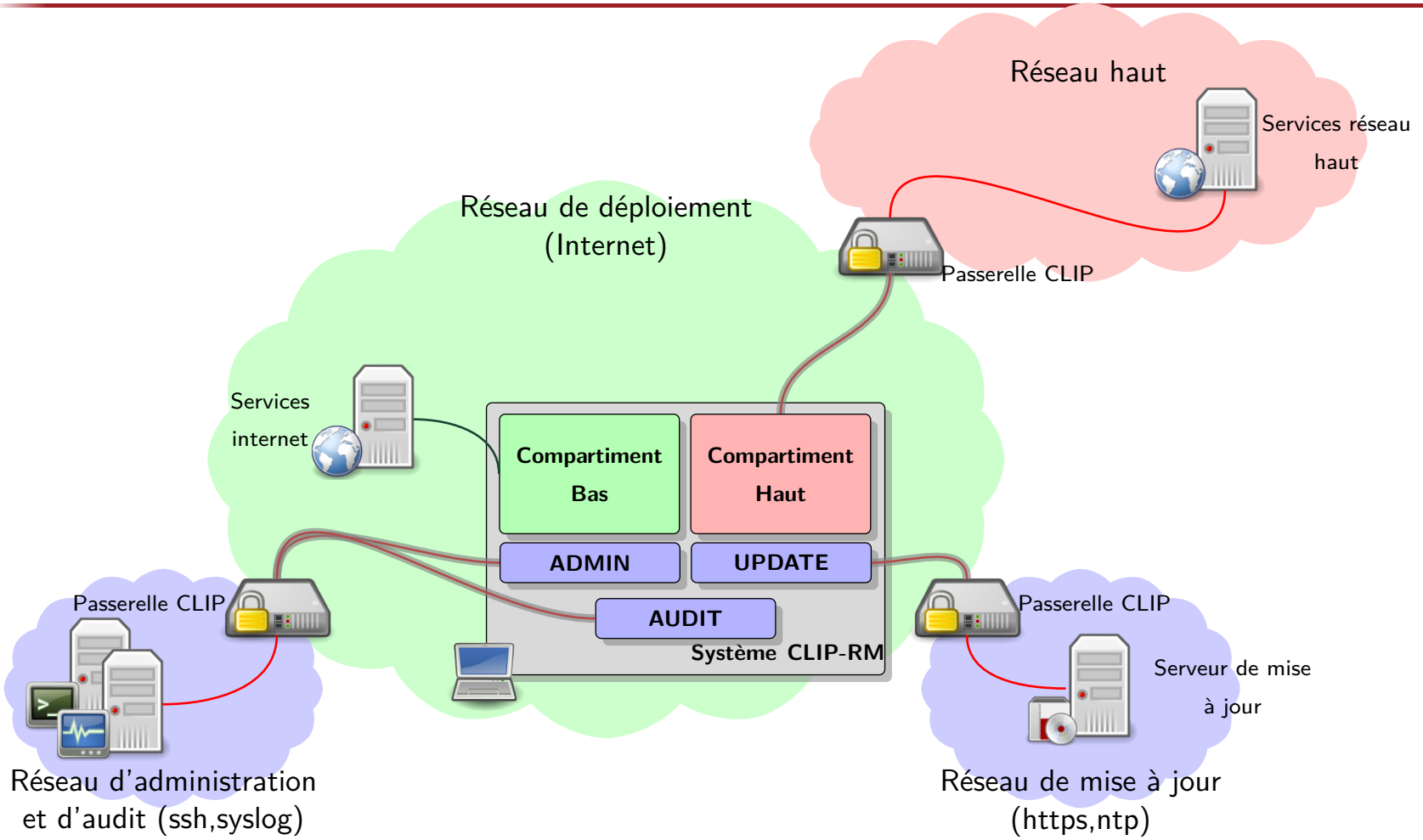
CLIP dans son environnement



CLIP dans son environnement



CLIP dans son environnement



Principes de sécurité

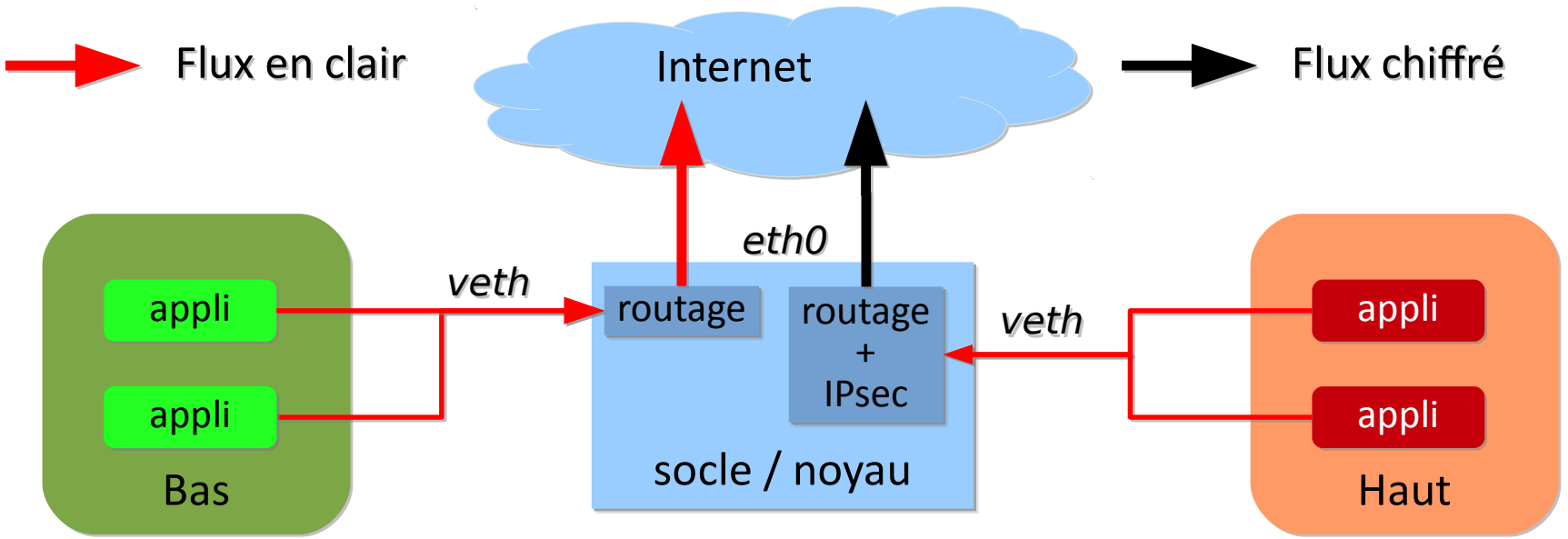
Cloisonnement : points délicats

Points durs

Les *containers* ne cloisonnent pas
"magiquement" certaines opérations

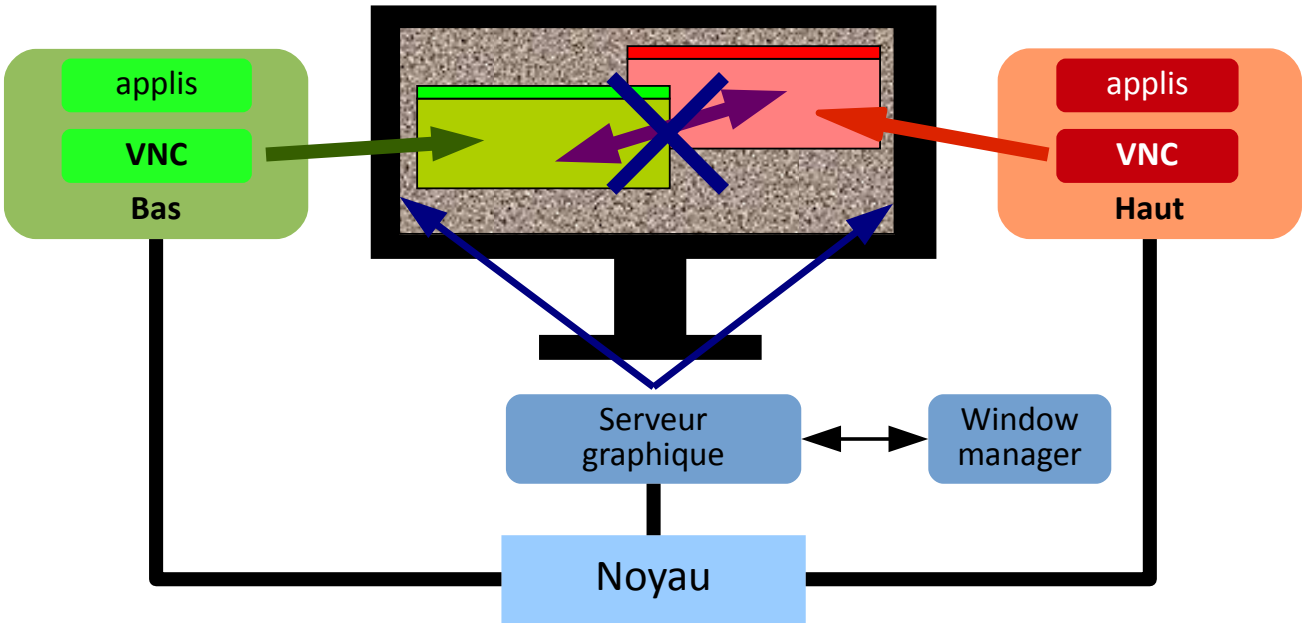
- Accès au réseau : une seule interface réseau
- Affichage : serveur X11 peu cloisonné par défaut
- Gestion des supports amovibles
- Accès au matériel en général

Cloisonnement réseau



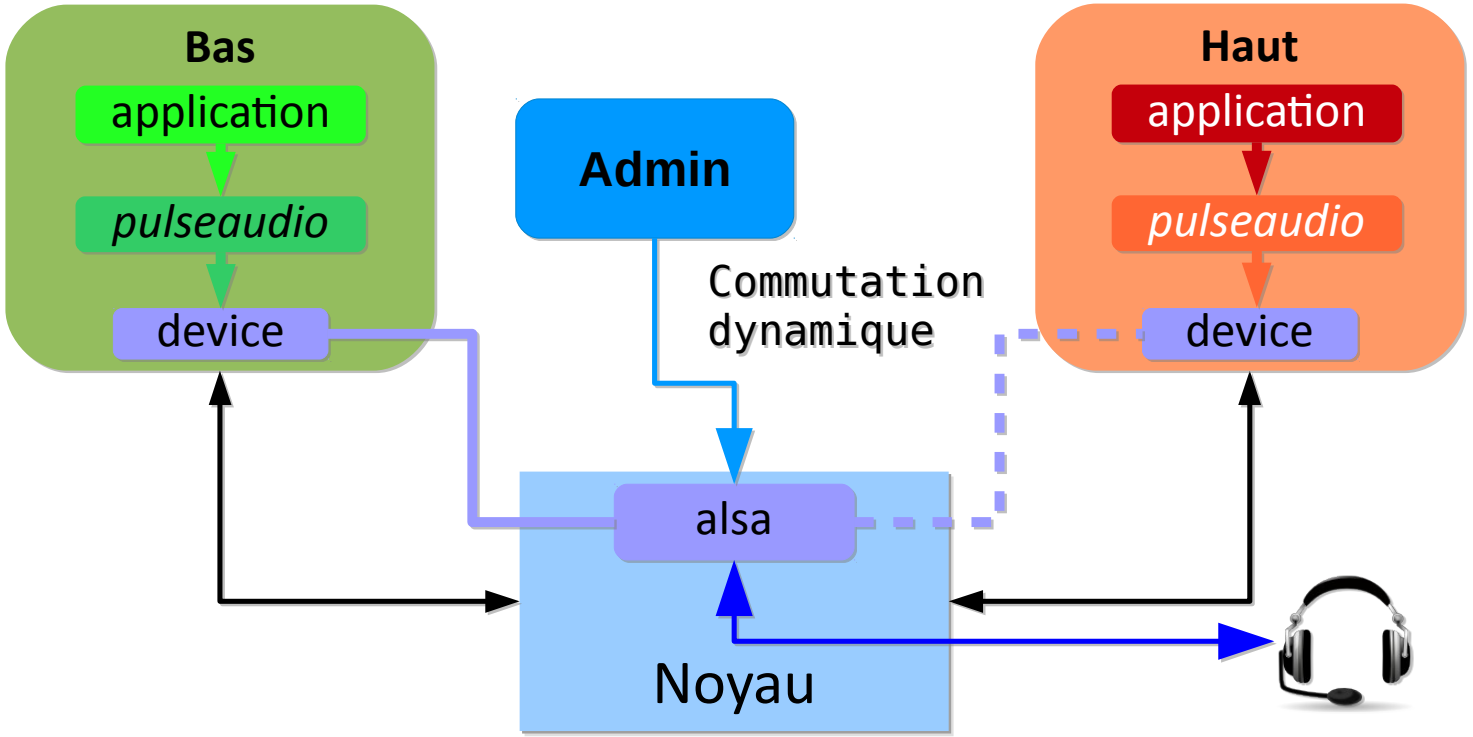
- Chaque cage ne voit qu'une interface virtuelle (*veth*) dédiée
- Le socle assure le routage entre interfaces virtuelles et réelles
- Règles *iptables* et IPsec propres à chaque cage

Cloisonnement graphique



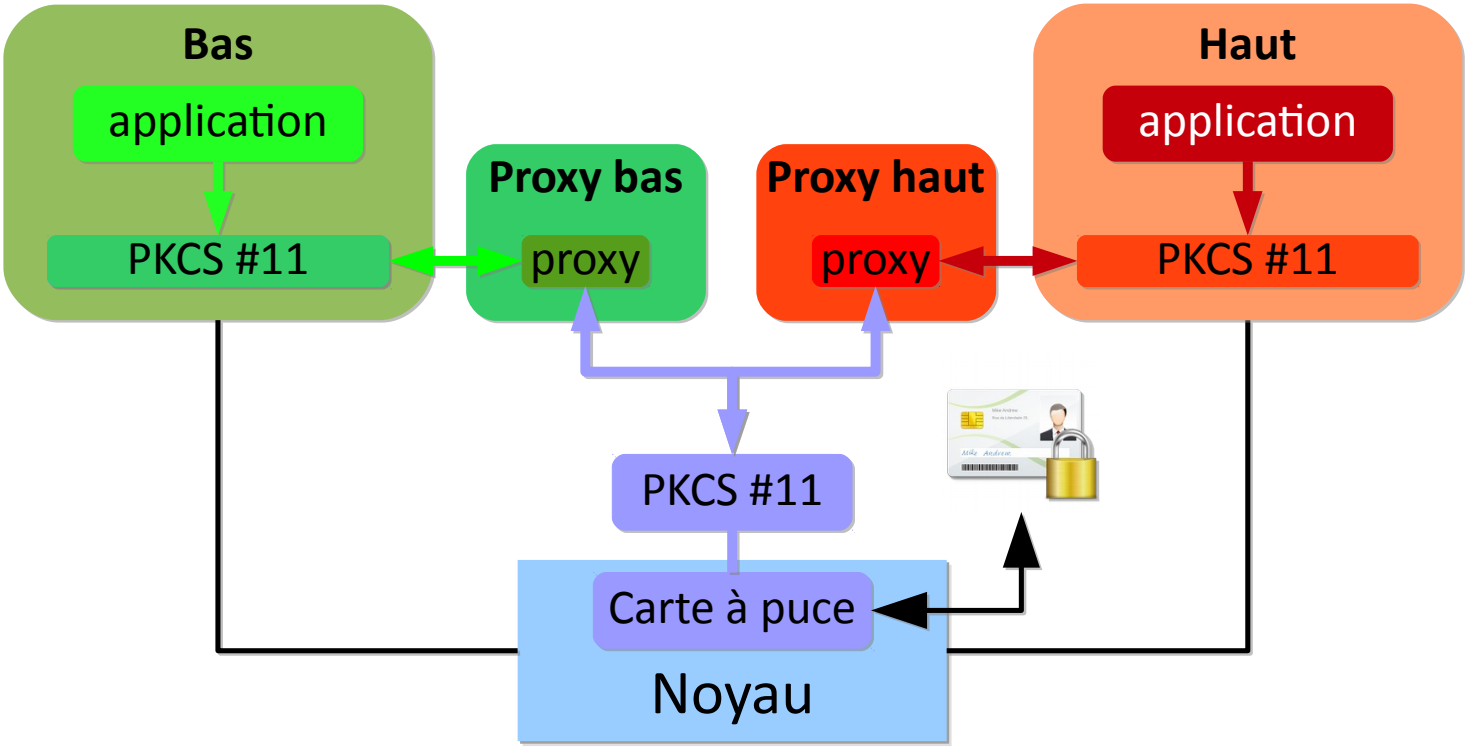
- Serveur X11 isolé et peu privilégié, affichage des cages via VNC
- Extension X11 pour cloisonner les *viewers* VNC et afficher le niveau via le *Window Manager*

Accès au matériel : périphériques dédiés



- Un périphérique est dédié à une cage *à un moment donné*
- Ex : commutation dynamique de la carte son

Accès au matériel : utilisation de proxys



- Filtrage par un proxy à haut niveau de sécurité
- Accès simultané à un périphérique "compartimentable"

Supports amovibles

- Clés USB signées
  - Affectées à un niveau particulier, signature dans une partition dédiée
  - Chiffrement optionnel
- Clés USB *lambda*
  - Montage autorisé dans le niveau choisi par l'utilisateur
  - Lecture-écriture au niveau bas, mais lecture seule au niveau haut
- CD, DVD
  - Uniquement en lecture, montage au choix de l'utilisateur

Principes de sécurité

Échanges entre cages

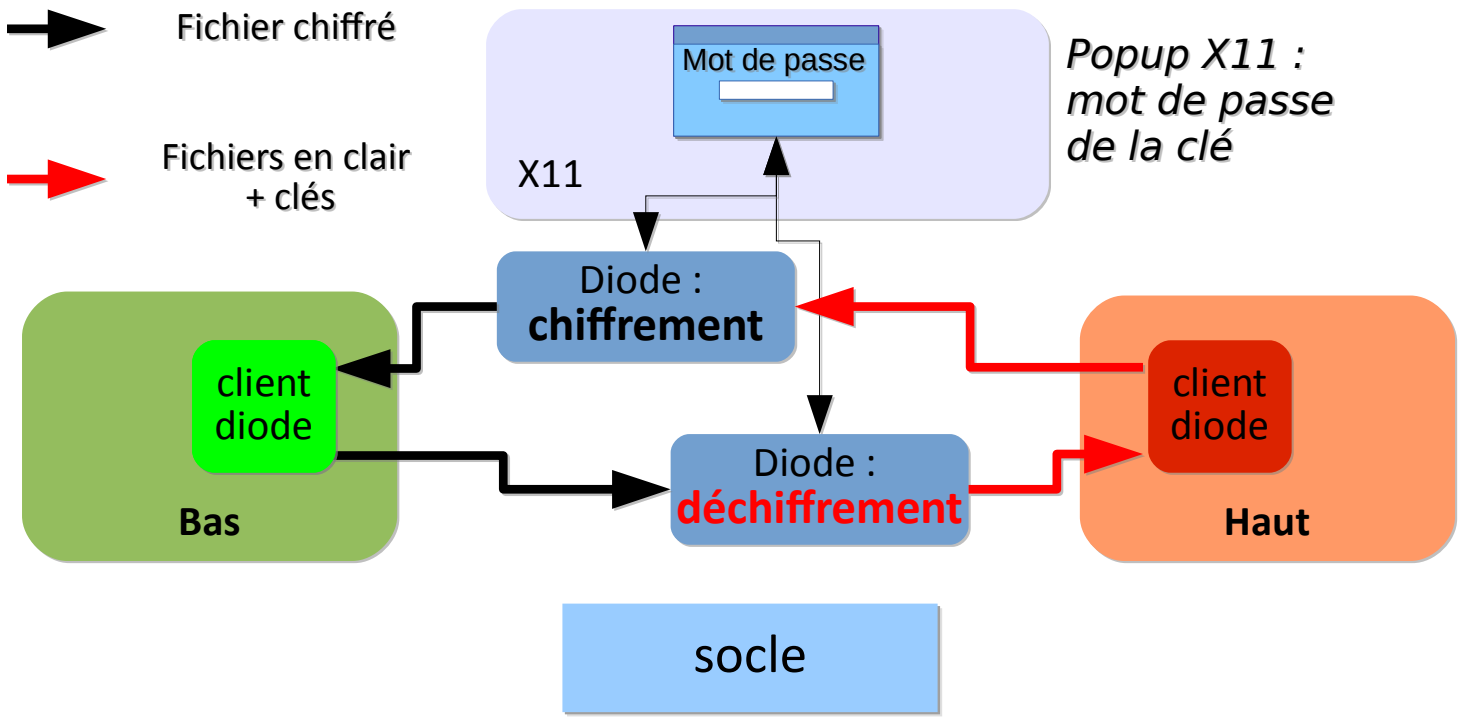
Échanges entre cages

Échanges autorisables via des sockets UNIX

- Remontée de logs : collecte *syslog* centralisée
- Délégation de privilège à un démon hors-cage
- Diode montante : transfert de fichiers bas vers haut
- "Diodes" crypto : chiffrement en coupure

Exemple : diodes crypto

- Chiffrement de fichiers de haut en bas
- Déchiffrement de bas en haut



Principes de sécurité
Durcissement et réduction de privilèges

Mesures de durcissement "classiques"

Gestion mémoire : PaX

- Principe $W \oplus X$ (quasi-)systématique en mémoire
- Randomisation mémoire

Chaîne de compilation "durcie"

- *stack-protector*, *relro* systématiques
- Tous les exécutables compilés *Position Independent*

Complexifie les attaques, sans les bloquer dans l'absolu

Réduction et séparation de privilèges

Pas de compte *root* interactif sur le poste

- Comptes administrateur et auditeur non privilégiés et "encagés"
- Privilèges limités à la modification de certains fichiers
  - Pas de *sudo vi ...*

Limitation des privilèges *root* des services

- LSM spécifique pour attribuer les capacités nécessaires à chaque exécutable
- Approche de type *secure-level* : opérations interdites après le démarrage (modules, écriture à la racine,etc.)

Principe W⊕X sur les fichiers

On ne peut pas exécuter un fichier qu'on a créé ou modifié

- Au moins une des options de montage :
  - *ro* : montage en lecture seule
  - *noexec* : exécution de binaires interdite
- Complété par une adaptation des interpréteurs de scripts, etc.
  - Nouvelle option *open(..., O_MAYEXEC)*

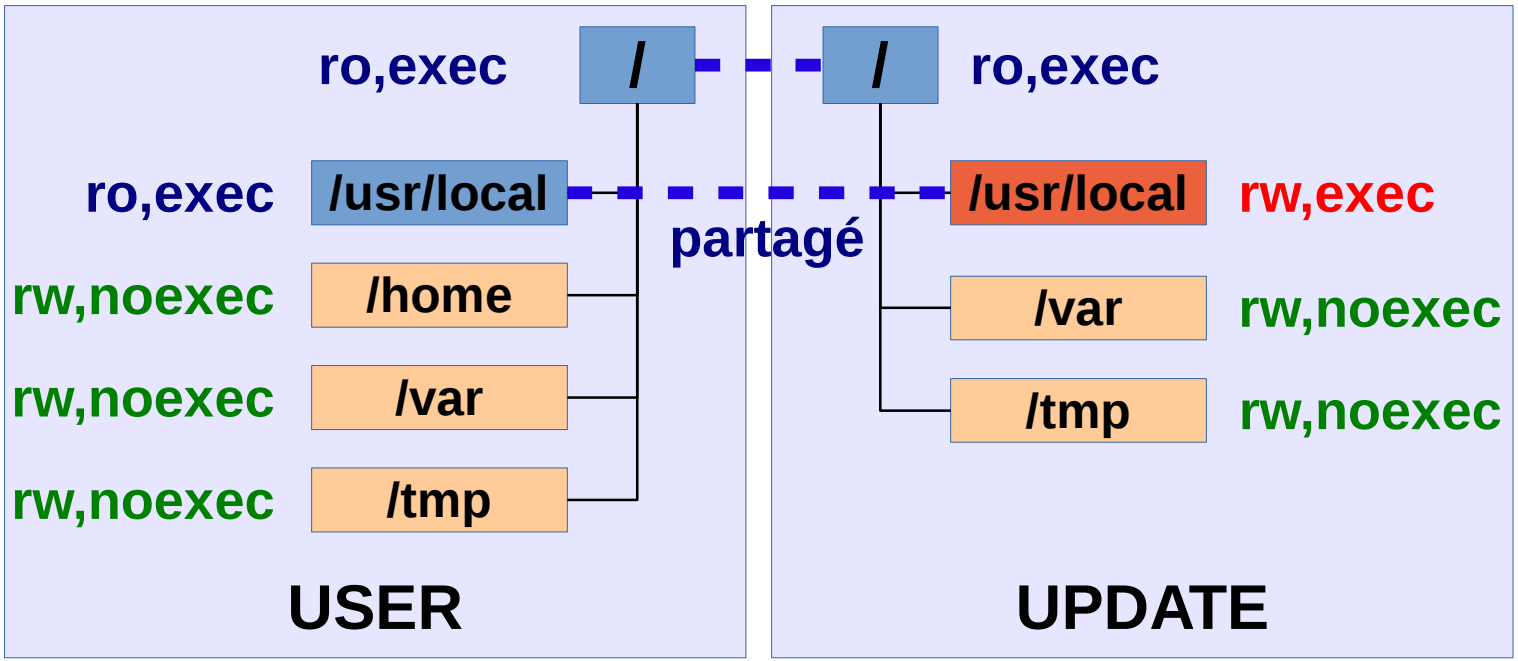
Gain obtenu : un attaquant ne peut pas importer ses outils d'exploitation pour pérenniser ou étendre une attaque

W⊕X sur les fichiers : mises à jour

Partage de certains montages entre USER et UPDATE

- W⊕X strict dans USER
- Accès en écriture à */usr/local* dans UPDATE

Les autres montages sont dédiés à chaque cage



Utilisation de la cryptographie

- Chiffrement de disque
  - Chiffrement global du système
  - Jeu de partitions chiffrées par utilisateur
  - Aucune écriture rémanente en clair après *log-out*
- Chiffrement réseau
  - IPsec / IKEv2 pour prolonger les cages
  - TLS / SSH pour la protection applicative
- Diode crypto
  - Échanges avec le monde hors-CLIP depuis le niveau haut

Principes de sécurité
Mises à jour sécurisées

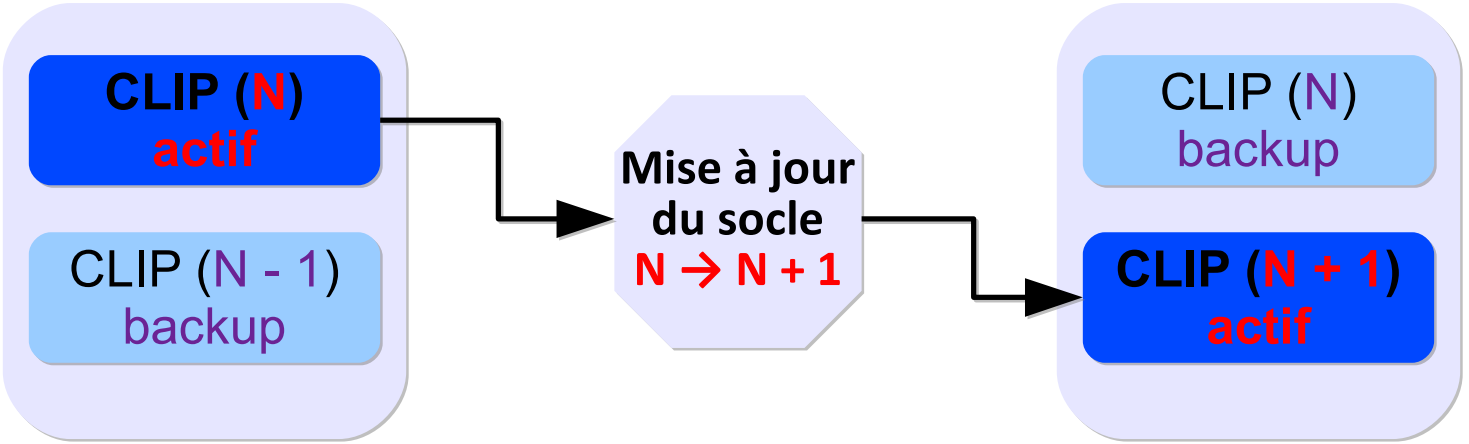
Mises à jour dans CLIP

Double objectif : sécurité et fiabilité

- Mises à jour entièrement automatiques
  - Pas d'interactions utilisateur, service en tâche de fond
  - Sauf redémarrage pour mises à jour de paquetages essentiels
- Authenticité et intégrité des mises à jour
  - Service de mise à jour isolé (cage dédiée)
  - Téléchargement à travers un tunnel IPsec
  - Signature cryptographique de chaque paquetage
    - Deux niveaux de signature : développeur / validateur

Mises à jour robustes

- Forte résistance aux interruptions et erreurs
- Opération "atomique" et réversible
  - Conservation et réinstallation au besoin des paquetages
- Deux jeux de partitions
  - Une partition *active*, une de *backup* en dernier recours



Évaluation du modèle, limites et pistes à creuser

Des mesures de sécurité complémentaires entre elles

L'attaquant a du mal à prendre pied dans le système

- Durcissement mémoire, options de compilation, minimalisation

Il ne prend pied que partiellement

- Confinement dans un compartiment logiciel

Il a du mal à se maintenir

- Impossibilité d'importer des exécutables
- Mise à jour automatique et sécurisée

Évaluation du modèle, limites et pistes à creuser
Limites du modèle

Méthode de cloisonnement : compromis assumé

Un cloisonnement trop haut niveau ?

- On reste tributaire du noyau Linux plutôt que d'un hyperviseur
- Mais on économise des ressources
- Et on ne réinvente pas la roue pour les échanges entre cages

Ou trop bas niveau ?

- Découpage à gros grain, SELinux offrirait plus de souplesse
- Mais aussi plus de risque d'erreur...

Limitations

Peu de mesures de sécurité "applicative"

- À l'intérieur d'une cage, sécurité quasi "standard"
- Protéger la cage contre le navigateur ? Le navigateur contre lui-même ?
- Mécanismes complémentaires : SELinux / RBAC / Landlock

Une résistance partielle à l'exécution de code arbitraire

- Protection à l'état de l'art contre l'injection de code
- Mais peu de mesures contre la *réutilisation de code*
- Le ROP / JOP est devenu la norme des *exploits*

Limitations (bis)

Capacité de détection limitée

- Remontée de logs et analyse distante possible
- Mais une analyse "intelligente" locale serait souhaitable
  - Surtout pour des usages en mobilité

Peu de protection contre les attaques "physiques"

- Pas/peu d'intégrité du disque
- Protection contre les *rogue peripherals* limitée

Évaluation du modèle, limites et pistes à creuser

Quelques idées à creuser

A better MAC

Rendre les primitives MAC utilisables

- SELinux ultrapuissant, mais utilisé uniquement avec une *refpolicy* générique et invérifiable
- Défi : générer des politiques adaptées et vérifiées

Compléter le modèle MAC

- Le noyau ne fait pas tout : *X11*, *dbus*, *udev / systemd* ...
- Défi : avoir une politique de sécurité entièrement cohérente
- Défi++ : qui intègre le navigateur...

Le moment de réinventer la roue ?

L'*uid* reste au coeur du modèle de sécurité de tous les OS

- Les couches MAC sont des bricolages complémentaires
- La majorité des usages sont mono-utilisateur
- Cf. décalage entre les modèles de l'OS et du navigateur

Quel modèle de sécurité pour le matériel ?

- Au-delà d'un empilement d'extensions...
- Des axes d'effort assez clairs en cas de remise à plat :
  - Contrôle d'exécution : garanties sur le CFG
  - Contrôle d'accès des périphériques

Constats et regrets

Regret 1 : peu de travaux de recherche transposés

- Il serait tentant d'intégrer plus de travaux de recherche dans CLIP
- Mais quasiment rien n'est publié / documenté / intégrable / maintenu

Regret 2 : essouflement de la sécurité OS / logiciel

- Pas de concept novateur depuis longtemps
- Le boulet creuse l'écart avec la cuirasse : ex. ROP/JOP
- Tout le monde est parti faire des hyperviseurs ou du javascript ?

Conclusion

Conclusion

- Un OS présentant des propriétés de sécurité intéressantes
  - Approche pragmatique : défense en profondeur plutôt que barrière inviolable
- Pas une solution sur étagère, mais presque
  - Facilement adaptable à plusieurs cas d'usage pertinents
- Après dix ans de maturation
  - Amélioration progressive des fonctionnalités, sans remise en cause de l'architecture de sécurité
- Rendu possible par des projets tiers
  - *PaX/grsecurity*, *vserver*, *gentoo hardened*...

Merci