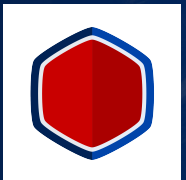
Sécurité système

Rémi Dubois • STI 2011



TLP:CLEAR

Diffusion non limitée

v.20230403

Rémi Dubois

- Ingénieur diplômé STI – ENSIB 2011
- Titre ESSI – ANSSI 2019
- Ingénieur SSI – cybersécurité & Responsable de SOC

Tour de table

- Qui êtes-vous ?
- Quelle est votre formation ?
- Quel type de poste occupez-vous ?
- Qu'attendez-vous de ce module ?

Plan des cours

TLP:CLEAR

Sécurité système

Principes généraux

TLP:CLEAR

Diffusion non limitée

Plan

TLP:CLEAR

Critères de sécurité recherchés

Il s'agit des 4 critères classiques :

- Confidentialité
- Intégrité
- Disponibilité
- Traçabilité / auditabilité

Moindre privilèges

Définition

Ne disposer que des droits strictement nécessaires à l'exécution, et rien de plus.

Corollaires :

- Minimalisation (KISS) : réduction de la surface d'attaque
  - N'installer que ce qui est nécessaire
  - N'activer et n'exposer que ce qui est nécessaire
- Restriction au besoin d'en connaître
  Un des deux principes du classifié de défense

Application

- Ne pas utiliser un compte d'administration au quotidien
  `Sudo`er ou `Administrateur`, ou pire, `root` ou `SYSTEM`
- Utiliser un poste dédié à l'administration
- Configurer précisément et durcir le système
  Montage des partitions, AppArmor, SELinux…
- Diviser, cloisonner et isoler les services

Défense en profondeur

Définition

défense globale et dynamique, coordonnant plusieurs lignes de défense couvrant toute la profondeur du système.

Mémento de la DCSSI, 2004

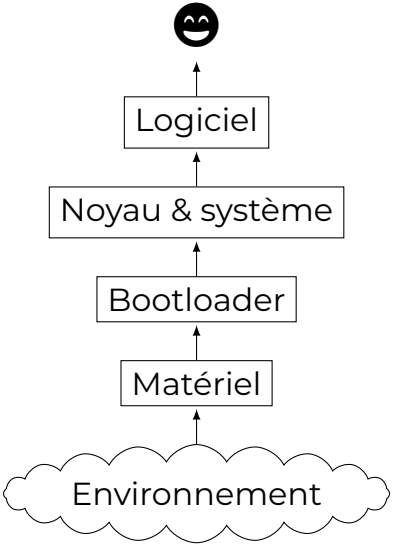
- Principe de construction des forteresses
- Chaque ligne de défense ralentit l'adversaire
- Classique en matière industrielle (multiples « barrières »)

Défense en profondeur

TLP:CLEAR

Application

- Appréhender l'ensemble du système
- Son environnement (physique et réseau)
- Son matériel
- Son noyau et couches basses
- Ses logiciels et couches hautes
- Sa maintenance



L'attaquant peut-il ?

- Accéder sans limite au matériel ?
  Matériel sans surveillance et sans verrouillage :
  - douanes dans certains pays
  - casier à l'entrée d'une salle de réunion
- Analyser *a posteriori* le contenu du support de stockage ?
  Disque dur accessible sans chiffrement
- Maîtriser la chaîne de démarrage ?
  SecureBoot, options du noyau modifiables
- Injecter du code dans le noyau ?
  Drivers

Application

- Filtrer les flux réseau globalement **et** localement
- Diversifier : utiliser plusieurs types de pare-feu, moteurs antiviraux, logiciels socles (DNS, mail…)
- Journaliser : la confiance n'exclue pas le contrôle
- Être à jour : installer systématiquement les correctifs de sécurité
- Un peu de paranoïa : **tout** ce qui n'est pas normal **est** anormal
- Ne jamais faire confiance à l'utilisateur

Questions?

Sécurité système

Installer un système sécurisé

TLP:CLEAR

Diffusion non limitée

Plan

TLP:CLEAR

1. Orientations techniques
2. Partitionnement
3. Configuration noyau (`sysctl`)

Choisir le système en fonction :

- des services qu'il va héberger
  Linux = polyvalent mais gros, *BSD plus minimaliste ?
  Disponibilité des fonctionnalités souhaitées ?
- du besoin d'avoir un support commercial (RHEL, SLES…)
  parfois nécessaire pour le support des logiciels (Oracle…)
- de la durée d'utilisation et la capacité de le maintenir à jour
  RIP CentOS et bof pour *BTW I use Arch*
- de la capacité à l'administrer !

Orientations techniques

Configuration matérielle

- Protéger le BIOS / UEFI
  Mot de passe administrateur ?
- Privilégier UEFI : SecureBoot
- Mode 64 bits
- Bit Intel VT-x / AMD V uniquement si virtualisation nécessaire

Configuration système

- Partitionner finement
- N'installer que le **strict minimum** nécessaire
  Pas d'interface graphique sur un serveur !
- Durcir les options noyau (`sysctl`)
- Durcir le système (règles `sudoers` granulaires…)
- Durcir les services (empêcher les connexions `root` sur SSH…)
- Filtrer les accès (Netfilter `iptables` / `nftables`)
- Mot de passe `root` complexe et spécifique à ce système
  Au coffre-fort ou dans un gestionnaire de mot de passe

Chiffrement de disque

- Utiliser un secret distinct (*passphrase* ou *token*)
  LUKS / LUKS2 (Linux) ou BitLocker (Windows **Pro**)
- Impératif pour une **station nomade** !
- Envisageable pour une station fixe
  Permet d'éviter de disséminer des documents sensibles lors de la mise au rebus
- Plus discutable pour un serveur
  Comment saisir la *passphrase* ?
- Prévoir un clé de séquestre (accident, réquisition judiciaire…)
  Au coffre-fort ou dans un gestionnaire de mot de passe

Partitionnement

Partitionnement I

Point de montage	Options	Description
`/`	<sans option>	Racine, contient le reste
`/boot`	`nosuid,nodev,noexec` (`noauto` optionnel)	Contient le noyau et le boot-loader, pas d'accès après démarrage sauf mise à jour
`/tmp`	`nosuid,nodev,noexec`	Fichiers temporaires, ne doit pas contenir d'exécutables. Nettoyé après redémarrage, préférer du `tmpfs`
`/proc`	`hidepid=2`	Informations sur les processus et le système

Partitionnement II

TLP:CLEAR

Point de montage	Options	Description
`/home`	`nosuid,nodev,noexec` (`ro` optionnel)	Répertoires utilisateurs. `ro` si non utilisé. `noexec` à discuter sur station de travail
`/usr`	`nodev`	Majorité des utilitaires (exécutables et bibliothèques)
`/var`	`nosuid,nodev,noexec`	Fichiers variables pendant la vie du système (mails, BDD...)
`/var/log`	`nosuid,nodev,noexec`	Journaux du système
`/var/tmp`	`nosuid,nodev,noexec`	Fichiers temporaires conservés après extinction

Partitionnement III

TLP:CLEAR

Point de montage	Options	Description
`/opt`	`nosuid,nodev` (`ro` optionnel)	Packages additionnels. `ro` si non utilisé
`/srv`	`nosuid,nodev` (`noexec,ro` optionnels)	Fichiers servis par un service

D'après les Recommandations de sécurité relatives à un système GNU/Linux, ANSSI

Nota : utiliser correctement le système de fichier et se référer au *Filesystem Hierarchy Standard*, `hier(7)`

- `nodev` empêche de jouer avec les **fichiers spéciaux** pour accéder sans droit ni titre à une partition autrement protégée (avec `mknod` utilisé par exemple sur une clé USB) :

```
brw-rw---- 1 root disk     8,   0 18 mars  16:29 sda
brw-rw---- 1 root disk     8,   1 18 mars  16:29 sda1
brw-rw---- 1 root disk     8,   2 18 mars  16:29 sda2
brw-rw---- 1 root disk     8,   3 18 mars  16:29 sda3
```

- `/var/log` peut vous exploser à la figurer et bloquer le démarrage d'applications (comme X.org)

Configuration noyau (`sysctl`)

Configuration noyau (`sysctl`)

TLP:CLEAR

Se référer aux Recommandations de sécurité relatives à un système GNU/Linux de l'ANSSI :

- pour la pile réseau, durcir les configuration IPv4 et IPv6 face à des paquets peu commun ou non standards
  ANSSI : serveur sans routage et avec adresse IPv6 statique
  Pas d'IPv6 : `net.ipv6.conf.all.disable_ipv6 = 1`
- pour le reste du système : paramétrage de la mémoire, des processus, du système de fichiers...

Configuration noyau (sysctl) - Options rése

TLP:CLEAR

```
# Pas de routage entre les interfaces
net.ipv4.ip_forward = 0
# Filtrage par chemin inverse
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
# Ne pas envoyer de redirections ICMP
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0
# Refuser les paquets de source routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
# Ne pas accepter les ICMP de type redirect
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
# Loguer les paquets ayant des IPs anormales
net.ipv4.conf.all.log_martians = 1
```

```
# RFC 1337
net.ipv4.tcp_rfc1337 = 1
# Ignorer les réponses non conformes à la RFC 1122
net.ipv4. icmp_ignore_bogus_error_responses = 1
# Augmenter la plage pour les ports éphémères
net.ipv4.ip_local_port_range = 32768 65535
# Utiliser les SYN cookies
net.ipv4.tcp_syncookies = 1
# Désactiver le support des "router solicitations"
net.ipv6.conf.all.router_solicitations = 0
net.ipv6.conf.default.router_solicitations = 0
# Ne pas accepter les "router preferences" par "router
   advertisements"
net.ipv6.conf.all.accept_ra_rtr_pref = 0
net.ipv6.conf.default.accept_ra_rtr_pref = 0
# Pas de configuration auto des prefix par "router advertisements"
net.ipv6.conf.all.accept_ra_pinfo = 0
net.ipv6.conf.default.accept_ra_pinfo = 0
```

```
# Pas d'apprentissage du routeur par défaut par "router
   advertisements"
net.ipv6.conf.all.accept_ra_defrtr = 0
net.ipv6.conf.default.accept_ra_defrtr = 0
# Pas de configuration auto des adresses à partir des "router
   advertisements"
net.ipv6.conf.all.autoconf = 0
net.ipv6.conf.default.autoconf = 0
# Ne pas accepter les ICMP de type redirect
net.ipv6.conf.all.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0
# Refuser les packets de source routing
net.ipv6.conf.all.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0
# Nombre maximal d'adresses autoconfigurées par interface
net.ipv6.conf.all.max_addresses = 1
net.ipv6.conf.default.max_addresses = 1
```

TLP:CLEAR

```
# Désactivation des SysReq
kernel.sysrq = 0
# Pas de core dump des exécutables setuid
fs.suid_dumpable = 0
# Interdiction de déréférencer des liens vers des fichiers dont
# l'utilisateur courant n'est pas le propriétaire
# Peut empêcher certains programmes de fonctionner correctement
fs.protected_symlinks = 1
fs.protected_hardlinks = 1
# Activation de l'ASLR
kernel.randomize_va_space = 2
# Interdiction de mapper de la mémoire dans les adresses basses
   (0)
vm.mmap_min_addr = 65536
```

Configuration noyau (sysctl) - Options syst

TLP:CLEAR

```
# Espace de choix plus grand pour les valeurs de PID
kernel.pid_max = 65536
# Obfuscation des adresses mémoire kernel
kernel.kptr_restrict = 1
# Restriction d'accès au buffer dmesg
kernel.dmesg_restrict = 1
# Restreint l'utilisation du sous système perf
kernel.perf_event_paranoid = 2
kernel. perf_event_max_sample_rate = 1
kernel.perf_cpu_time_max_percent = 1
```

Questions?

Sécurité système

Cloisonnement et isolation des processus

TLP:CLEAR

Diffusion non limitée

Plan

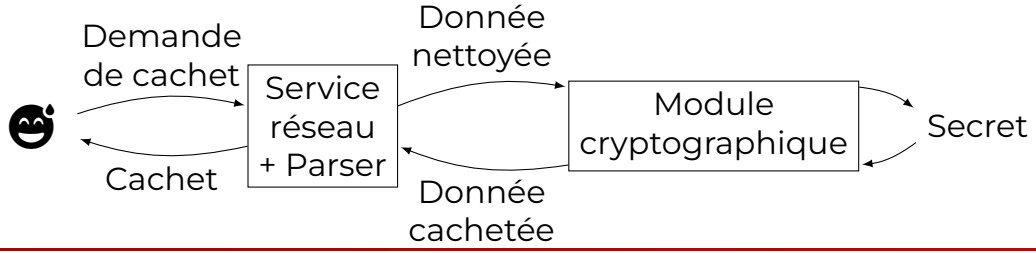
TLP:CLEAR

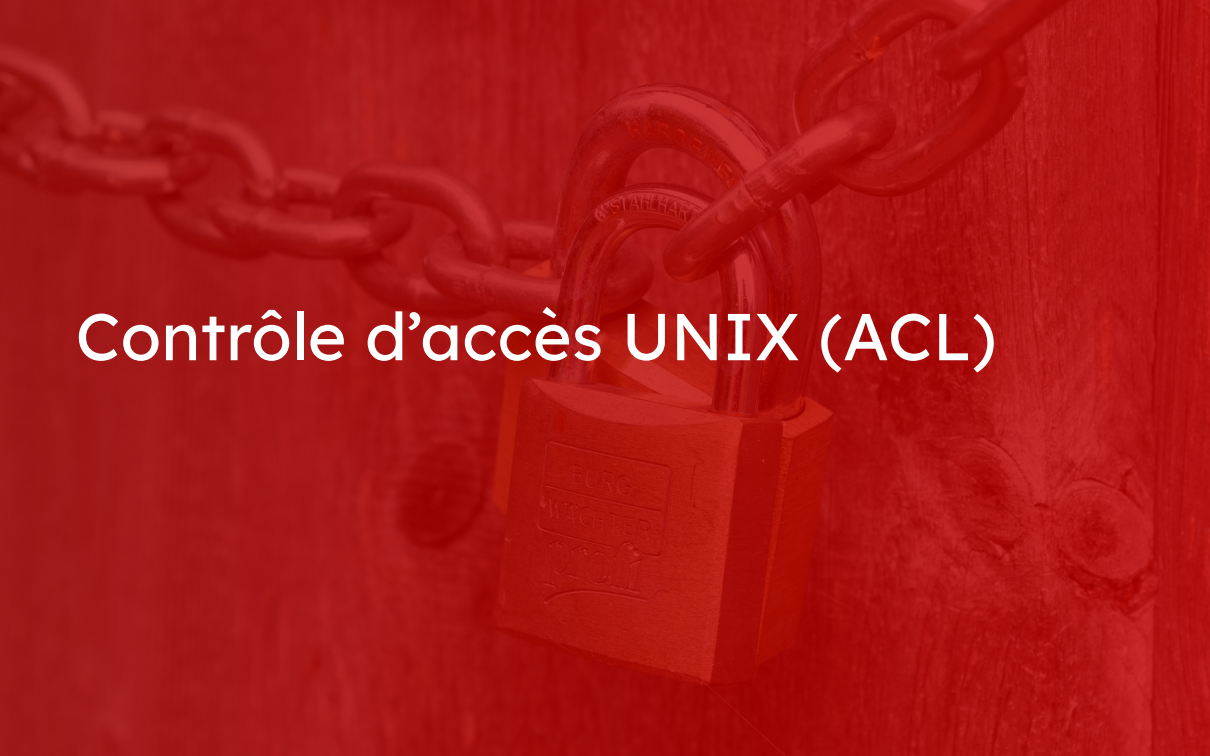
1. Contrôle d'accès UNIX (ACL)
2. Mécanismes anciens
3. Cgroups et *Namespaces* Linux
4. Conteneurs et virtualisation

Pourquoi cloisonner ?

- Séparer les fonctionnalités d'une application
- Isoler des actions potentielles dangereuses
- Limiter les privilèges requis (ports privilégiés…)
- Garder les secrets… secrets

Exemple : service de cachet serveur





Contrôle d’accès UNIX (ACL)

Rappels

Système de contrôle d'accès discrétionnaire (DAC) où tout est fichier :

- Utilisateur propriétaire (`u`)
- Groupe propriétaire (`g`)
- Autres utilisateurs (`o`)
- Lecture (`r`)
- Écriture (`w`)
- Exécution / traversée (`x`)

Ne pas oublier sur Linux :

- ACL étendues (`getfacl` / `setfacl`)
- On peut ainsi (un peu) s'orienter vers du contrôle d'accès par rôles (RBAC)

Concept d'utilisation

- Utilisateur spécifique par processus/fonctionnalité
- Groupe commun
- Contrôle plus fin avec des ACL étendues

Exemple : Serveur web (**`httpd:http`**) et PHP FPM (**`php:http`**)

```
srw-rw---- 1 httpd http    0  5 mars  16:21 /run/php-fpm.sock
-r-------- 1 php   http 3272 13 févr.  2021 /etc/myapp/privkey.pem
```

Exemple : Répertoire de numérisation pour photocopieur

```
ls -lR /srv/partage
drwxrwx---+ 7 root secretariat 4096 15 janv.  2019 Secrétariat

/srv/partage/Secrétariat
total 38
drwxrwx-w-  2 root secretariat 4096 15 janv.  2019 Numérisation
```

```
getfacl /srv/partage/Secrétariat
# file: srv/partage/Secrétariat
# owner: root
# group: secretariat
user::rwx
group::rwx
other::---
user:photocopieur:--x
```

DAC souffre de nombreuses limitations :

- les utilisateurs ont un contrôle complet de leurs ressources (CRUD + nouveaux droits)
- les programmes ont les mêmes droits que leur utilisateur

→ Si un programme en `UID 0` est compromis, le système entier est compromis

Security-Enhanced Linux apporte un système de contrôle d'accès mandataire (MAC) où le noyau applique une politique de contrôle complémentaire.

SELinux utilise des contextes de sécurité composés :

- d'une identité (`_u`, propriétaire du contexte)
- d'un rôle (`_r`)
- d'un domaine pour les sujet (utilisateurs ou processus) ou d'un type pour les objets (ressources passives) (`_t`).

Le contexte d'un objet peut être vu avec `ls -Z` :

```
drwxr-xr-x root root system_u:object_r:etc_t  /etc
```

Écrire une politique de sécurité est **difficile**.
Certaines distributions appliquent par défaut une politique SELinux, comme RHEL qui se base sur le FHS.

Mécanismes anciens

chroot / jail

chroot (appel système et commande) simule une autre racine à l'intérieur du système de fichiers

- seuls les fichiers sous la nouvelle racine seront visibles
- nécessite de recréer une arborescence FHS (**/**, **/dev**, **/proc**, **/sys**…), et d'exposer les bibliothèques nécessaires
- nécessite des privilèges élevés pour être utilisé (ou **CAP_SYS_CHROOT**), mais utiliser **root** ne respecte pas le principe de moindre privilèges (→ **setuid**)
- *chroot is not and never has been a security tool* (Alan Cox)
- *This call [...] is not intended to be used for any kind of security purpose* (**chroot(2)**)

`chroot / jail`

`jail` est une alternative plus robuste sous FreeBSD avec :

- un appel système
- un jeu de commandes (**`jail`**, **`jls`** et **`jexec`**)

Un **`jail`** a aussi son propre environnement réseau (nom d'hôte et adresse IP.)

Faute de mieux, on a le programme **`firejail`** sous Linux, utilisant les *namespaces* et *seccomp-bpf*.

`setuid` et `setgid`

Ces deux appels systèmes changent l'utilisateur et le groupe principal du processus :

- fait par la plupart des services *sérieux* lancés en **`root`**
- sinon, écrire un service systemd adapté (**`systemd.exec(5)`**)
- y dédier un compte de service spécifique
- y dédier un groupe spécifique à un ensemble de services (cf. exemple ACL)

Pour limiter les privilèges requis sur certaines actions sensibles, Linux propose les **`capabilities(7)`**

seccomp(-bpf)

À l'origine, **seccomp(2)** est un appel système pour jeter ses droits et se restreindre à la manipulation de fichiers déjà ouverts (SECCOMP_SET_MODE_STRICT).

Effectuer un autre appel système que **exit**, **sigreturn**, **read** ou **write** conduit à se faire tuer par le noyau.

Le mode filtrer *Berkley Packet Filter* (SECCOMP_SET_MODE_FILTER) permet de préciser la liste des appels systèmes autorisés.

seccomp-bpf a ainsi été particulièrement utilisé par Chromium OS dont le code source peut servir de référence d'implémentation.

Cgroups et *Namespaces* Linux

La fonctionnalité Linux des *Control groups* (**`cgroups(7)`**) permet de :

- organiser les processus en groupes hiérarchiques
- limiter leurs ressources
- les superviser (**`systemd-cgtop`**)

Ils sont présentés à l'utilisateur comme un simple système de fichiers monté (**`/sys/fs/cgroup`**) : *tout est fichier*.

systemd crée par défaut un cgroup sous **`system.slice`** pour chaque service (→ **`systemd-cgls`**).

Les `(*_)namespaces(7)` permettent de cloisonner les applications sur plusieurs types de ressources :

Type	Fonctionnalité
Mount	Isolation des points de montage `chroot` en *beaucoup* mieux
UTS	Isolation du nom d'hôte et de domaine
IPC	Isolation des communications interprocessus (IPC System V, files de messages POSIX)
User	Isolation des identifiants utilisateurs et des groupes Permet d'avoir un `UID 0` sans être vraiment privilégié
PID	Isolation des identifiants des processus
Network	Isolation des ressources réseau (interfaces, ports…)
Cgroup	Isolation des Cgroups
Time	Isolation des horloges système

Conteneurs et virtualisation

Conteneurs

Avec les fonctionnalités précédentes, il est possible sur un même système hôte :

- de mettre en place des sous-systèmes Linux (conteneurs)
- d'isoler les processus et les ressources associées

Ces conteneurs peuvent être gérés :

- bas niveau avec les commandes LXC (`lxc-*`)
- avec `systemd-nspawn` / `systemd-machined` (`machinectl`)
- plus haut niveau avec Docker ou containerd pour faciliter la diffusion et le déploiement

Voir aussi les Recommandations pour la mise en place de cloisonnement système, ANSSI

Conteneurs ±

TLP:CLEAR

Avantages

- isolation des composants applicatifs au sein des conteneurs
- réduction des privilèges (`user_namespaces`…)
- scalabilité horizontale facilitée par la répétabilité du déploiement

Inconvénients

- complexité accrue de l'architecture applicative
  (quel conteneur héberge quoi ?)
- complexité accrue de l'administration (socle + gestionnaire)
- jonction (plus) forte entre administration système et développement (*devops*)
- est-ce adapté pour assurer une sécurité élevée ?
  (confiance dans l'isolation offerte par le noyau)

Virtualisation

TLP:CLEAR

Lorsqu'une segmentation plus importante ou des fonctionnalités spécifiques du noyau sont requises (comme le filtrage réseau), l'isolation par conteneurs ne suffit plus.

On peut alors employer la virtualisation où un **hyperviseur** (comme KVM, VirtualBox, Xen ou VMware) émule ou **partage les ressources physiques** (CPU, mémoire, stockage, I/O) de l'hôte.

Le matériel doit être configuré pour supporter la virtualisation (bits Intel VT-x ou AMD-V...)

Voir aussi les Problématiques de sécurité associées à la virtualisation des systèmes d'information, ANSSI

Virtualisation ±

Avantages

- accès aux fonctionnalités spécifiques du noyau
- isolation plus marquée, reposant sur des fonctionnalités du CPU
- performances accrues grâce aux ressources matérielles pouvant être dédiées

Inconvénients

- matériel dédié pour un hyperviseur type 1 (*bare metal*)
- possible complexité de l'architecture de virtualisation (cluster d'hyperviseurs, stockage SAN…)
- coût des potentielles licences (VMware…)
- est-ce adapté pour assurer une sécurité très élevée ? (confiance dans l'isolation offerte par l'hyperviseur)

Questions?

Sécurité système

Journalisation et centralisation

TLP:CLEAR

Diffusion non limitée

Plan

TLP:CLEAR

Pourquoi journaliser ?

TLP:CLEAR

- Aider à comprendre ce qui se passe pour débugger
- Assurer une traçabilité légale ou réglementaire
- Contrôler : la confiance n'exclue pas le contrôle
- Détecter et analyser les incidents de sécurité

La suite est librement inspirée des Recommandations de sécurité pour l'architecture d'un système de journalisation, ANSSI

Journaliser ?

Que journaliser ?

TLP:CLEAR

- Authentifications et élévations de privilèges
- Gestion des comptes, rôles (groupes) et privilèges
- Modification des stratégies de sécurité (dont journalisation)
- Accès aux ressources sensibles (RWX + suppression)
- Activité des processus ⚠ volumétrie
- Activité des systèmes (démarrage, modules noyau, matériel…)

- Événements de sécurité pare-feu

Comment journaliser ?

TLP:CLEAR

- Prévoir la fonctionnalité dans les applications
- Activité système, processus… : **`auditd(8)`**
- Fonctionnalité native ou **`syslog`**/**`journald`** + **`logrotate`**
- Identifier la source des journaux (application & système)
- Horodater ! (avec une même source de temps : NTP)
- Données structurées

Collecter et centraliser ?

Pourquoi collecter et centraliser ?

TLP:CLEAR

- Assurer le stockage dans la durée (plusieurs mois)
  Espace disque local probablement faible
- Assurer l'intégrité des journaux face à un attaquant
  - Contrôle de l'équipement ≠ contrôle des journaux exportés
  - Privilégier l'export en temps-réel !
- Faciliter la recherche avec un outil adapté (`grep` = )
- Corréler les événements de plusieurs sources (SIEM)

Comment centraliser ?

TLP:CLEAR

- Utiliser des protocoles sécurisés (confidentialité, intégrité, authentification) natifs ou avec TLS, à défaut SSH ou IPSec
- Durcir et cloisonner la plateforme de centralisation (+ MCS)
- Prévoir une partition dédiée (cf. principes généraux)
- Superviser la volumétrie peut vite déborder
- Préférer une base de données structurée et indexée à des fichiers à plat
  - comme Elasticsearch + Logstash
  - à défaut, `syslog-ng` / `rsyslog`

Comment l'exploiter ?

TLP:CLEAR

- Recherche des anomalies « évidentes »
  (utilisateurs ou horaires atypiques, erreurs)
- Superviser les scénarios de risques ou de menaces identifiés
  (analyse de risques)
- Leitmotiv : **tout** ce qui n'est pas normal **est** anormal !

Questions?

Sécurité système

Travaux dirigés

TLP:CLEAR

Diffusion non limitée

Plan

TLP:CLEAR

1. TD : Installer un système sécurisé
2. TD : Cloisonner et isoler
3. TD : Journaliser et centraliser

TD : Installer un système sécurisé

Installer un système sécurisé sur base Debian

- Chiffrement des disques
- Partitionnement et montage adaptés
- Durcissement noyau (**`sysctl`**)
- Durcissement OpenSSH (**`PermitRootLogin no`** au minimum)
- Filtrage Netfilter (⬇ SSH – ⬆ DHCP, DNS, HTTP[S])

→ Recommandations de l'ANSSI
- GNU/Linux (~~`nosuid,noexec`~~ sur **`/var`** pour Debian)
- (Open)SSH

TD : Cloisonner et isoler

Intégrer une application avec des composants isolés

- Nginx, uWSGI, Python et systemd
- Service HTTP (API JSON avec Flask) (`api.py`)
- Service cryptographique (calcul d'un HMAC-SHA256) (`crypto.py`)
- Communication par socket UNIX

→ Isolation des deux services Python avec systemd

Préparer le déploiement

1. Créer un groupe système (`groupadd(8)`)
2. Créer un compte de service (système, `useradd(8)`) pour chacun des deux scripts, en prenant garde à bien paramétrer les comptes (*home* et *shell*)
3. Déployer les scripts dans un répertoire conforme au FHS
4. Positionner le secret `key.bin` (paramétrage) dans un répertoire conforme au FHS et vérifier les permissions

Écrire le service systemd pour `crypto.py`

1. Créer un service systemd pour lancer **`crypto.py`**
   (voir `systemd.service(5)` et `systemd.exec(5)`)
   - le service doit utiliser son compte de service et son groupe
   - `SOCKET_PATH` et `KEY_PATH` doivent être paramétrées dans l'environnement
   - la socket devra être créée avec les bonnes permissions (*umask*) dans un répertoire géré par le service systemd sous **`/run`**
2. Charger et lancer le service :

```
systemctl daemon-reload
systemctl start <xyz>.service
```

3. Vérifier les permissions sur le répertoire sous **`/run`** et la socket
4. Tester le service :

```
echo "Hello world" | nc -U /run/.../xyz.sock | xxd
```

Écrire le service systemd pour `api.py`

1. Créer un service systemd pour lancer **`api.py`** avec uWSGI (`uwsgi-core(1)` et documentation Flask)
   - le service doit utiliser son compte de service et les groupes nécessaires pour communiquer avec le service cryptographique et Nginx
   - `SOCKET_PATH` doit être paramétrée dans l'environnement
   - une socket, accessible à Nginx, doit être créée dans un répertoire géré par le service systemd sous **`/run`**
2. Charger et lancer le service :

```
systemctl daemon-reload
systemctl start <xyz>.service
```

3. Vérifier les permissions sur le répertoire sous **`/run`** et la socket

Configurer Nginx

1. Configurer Nginx pour communiquer avec la socket uWSGI (voir documentation uWSGI)

```
location / { try_files $uri @cryptosrv; }
location @cryptosrv {
    include uwsgi_params;
    uwsgi_pass unix:/run/.../uwsgi.sock;
}
```

2. Redémarer Nginx et tester l'application :

```
systemctl restart nginx.service
http --json POST :/hmac data="Hello World"
```

TD : Journaliser et centraliser

Ajouter des fonctionnalités de journalisation centralisée

- Journaliser les requêtes et les opérations cryptographiques
- Conserver 7 jours localement
- Conserver 1 mois sur un serveur de centralisation

Modifier `crypto.py` pour journaliser

1. Installer **`python3-systemd`**
2. Importer **`logging`**, et **`journal`** depuis le module **`systemd`**
3. Configurer le *logger* racine (`logging.getLogger()`) pour le **`SYSLOG_IDENTIFIER='cryptopy'`** à partir du niveau **`logging.INFO`**
4. Utiliser **`logging.info()`** pour journaliser proprement les messages à la place des **`print()`**
5. Redémarrer le service et vérifier la journalisation :

```
systemctl restart cryptopy.service
journalctl -eu cryptopy.service
```

Modifier `api.py` pour journaliser

1. Configurer de même **`app.logger`** pour écrire vers systemd avec le **`SYSLOG_IDENTIFIER='apipy'`**
2. Journaliser la source (**`request.remote_addr`**) et le corps des requêtes ainsi que les HMAC correspondants générés
3. Recharger et redémarrer le service et vérifier la journalisation :

```
systemctl daemon-reload
systemctl restart apipy.service
journalctl -eu apipy.service
```

Relayer `journald` avec `rsyslog`

1. Configurer **`rsyslog`** pour :
   - enregistrer localement (**`omfile`**) les journaux de **`cryptopy`** et **`apipy`**, avec **`programname`** puis la date (les 10 premiers caractères du format **`date-rfc3339`**) dans le nom du fichier
   - et les relayer en RELP (**`omrelp`**) vers un second serveur
   - relayer de même les *facilities* **`auth`** et **`authpriv`**
2. Redémarrer **`rsyslog`** puis vérifier les journaux :

```
systemctl restart rsyslog.service
systemctl restart cryptopy.service apipy.service
cat /var/log/cryptosrv/*
```

Configurer le serveur de centralisation

1. Configurer **`rsyslog`** pour :
   - écouter en RELP (**`imrelp`**)
   - enregistrer les journaux dans un fichier (**`omfile`**) avec le nom du serveur source et la date dans le nom du fichier
2. Redémarrer **`rsyslog`** :
   ```
   systemctl restart rsyslog.service
   ```
3. Générer des journaux et vérifier leur centralisation

Pour aller plus loin

- Identifier les ressources sensibles et les surveiller avec `auditd`
- Mettre en place une pile Elasticsearch

→ Recommandations de l'ANSSI

- Journalisation
- Journalisation Windows

En cas de besoin :

remi@rdubois.fr