Processus d'autorisation `sudo` par projet et rotation automatique des bridges Tor

Document technique — Kali One (c3b) — WebTech3

- **Date** : 2026-10-15
- **Machine** : Kali One (Debian/Kali), utilisateur `c3b`, hôte `One`
- **Auteur** : DSH, agent local (pont Open WebUI ↔ DSH, `127.0.0.1:20130`)
- **Destinataire** : Pierre (c3b) — publication `webtech3.com`
- **Objet** : documenter de bout en bout (méthode → preuve → limite) la mise en place d'un **automatisme privilégié** : la rotation des bridges Tor d'entrée, et le cadre d'autorisation `sudo` qui l'accompagne.
- **Statut** : livrable vérifié, **installation en attente du mot de passe humain** (§7). Aucun droit `sudoers` n'a été modifié.
- **Publication** : ce document et le registre sont publiés sur le site WebTech3 — `https://webtech3.com/publications/` (vérifié en HTTPS le 15/10 : HTTP 200, `application/pdf`). Les empreintes de la version publiée sont dans `MANIFESTE.sha256`, déposé à côté des fichiers.

---

1. Résumé exécutif

Trois jours de travail sur la sécurisation de Tor n'avaient produit aucun résultat visible. Ce document explique **pourquoi**, **ce qui est réellement livré**, et **comment le prouver**.

Question	Réponse mesurée
Tor fonctionne-t-il aujourd'hui ?	**Oui.** `tor@default` `active`, `Bootstrapped 100%`, sortie `{"IsTor":true}`
Que corrige le livrable ?	Les bridges d'**entrée** : 2 des 4 en service sont **fermés**, dont un réinstallé à chaque rotation
Le correctif est-il testé ?	**Oui** : harnais **40 PASS / 0 FAIL**, manifeste **7/7 OK**, installation **4/4 empreintes**
Pourquoi n'est-il pas installé ?	`sudo` exige un mot de passe ; DSH tourne **sans TTY** (§7)
Un bridge peut-il être un nœud de **sortie** ?	**Non** — c'est une règle du protocole, pas un réglage (§6)

Le blocage n'est pas technique. C'est une porte que seul un humain peut ouvrir. Le reste est fini.

---

2. Contexte : pourquoi l'automatisation existait déjà

Contrairement à ce qu'on pouvait croire, la rotation automatique **existait** depuis le 24/09/2026 :

```
tor-securise-rotate.timer     OnCalendar=daily, RandomizedDelaySec=6h
tor-securise-rotate.service   ExecStart=/usr/local/bin/tor-securise-rotation
Dernière exécution : Thu 2026-10-15 01:23:46 CEST   status=0/SUCCESS
```

Elle a tourné le 15/10 à 01:23 et a **écrit** un nouveau jeu de bridges, puis redémarré Tor — qui a bootstrappé en 15 s. **Le problème n'était donc pas l'absence d'automatisation, mais sa qualité** : elle écrivait une liste non filtrée.

2.1 Le défaut mesuré

L'API `moat` (source des bridges) **ne filtre pas les bridges morts**. Mesure :

```
moat/circumvention/builtin → obfs4 : 7 candidats
  51.222.13.177:80      ← en production, sonde TCP 3/3 FERMÉ
  146.57.248.225:22     ← en production, sonde TCP 3/3 FERMÉ, port 22
  45.145.95.6:27015     ← OUVERT
```

La v1 installée prenait cette liste **brute**. Résultat : des bridges morts sont réinstallés à chaque passage, et Tor consomme 15 tentatives de connexion échouées par jour (`general SOCKS server failure`, de 01:23 à 09:52).

Ce qui a sauvé la connexion : sur 4 bridges, 3 étaient vivants. Tor sélectionne un bridge vivant et bootstrappe. **Le service n'est pas tombé — sa marge de sécurité, si.**

---

3. Méthode : le cycle en cinq temps

Le projet a suivi un cycle reproductible, applicable à tout futur automatisme privilégié :

```
1. MESURER      établir l'état réel par commande, jamais par supposition
2. MODÉLISER    écrire le correctif dans un dossier de travail isolé
3. ÉPROUVER     harnais de tests : chaque test doit pouvoir ÉCHOUER
4. SIMULER      rejouer l'installation sur une racine factice (/tmp)
5. PROUVER      empreintes sha256 + manifeste + test négatif du manifeste
```

3.1 Étape 1 — Mesurer

Outils et commandes employés (tous rejouables) :

```
sudo -n -l                                  # droits réellement accordés
systemctl list-timers tor-securise-rotate.timer --no-pager
journalctl -u tor@default --since '-12h'      # échecs de connexion bridge
grep -c '^Bridge' /etc/tor/torrc.d/50-bridges.conf
curl -s --socks5-hostname 127.0.0.1:9050 https://check.torproject.org/api/ip
bash scripts/tor-securise-rotation --list     # sonde TCP par bridge
```

3.2 Étape 2 — Modéliser

Le correctif v3 ajoute, par rapport à la v1 :

Ajout v3	**Effet**
Sonde TCP avant retenue	un bridge injoignable n'est jamais écrit
Liste noire	les morts récurrents sont rejetés et **nommés** dans la sortie
Élagage des morts en service	nettoie le drop-in existant, pas seulement les nouveaux

Ajout v3	Effet
Rejet des ports privilégiés	élimine les bridges sur port 22 (inutilisables par Tor)
Déduplication + tri stable	sortie reproductible, comparable d'une exécution à l'autre
Contrôle de vivacité APRÈS redémarrage	vérifie chaque bridge, pas seulement « Bootstrapped 100% »
Rollback automatique	si le bootstrap échoue, restaure les bridges précédents

3.3 Étape 3 — Éprouver

Le harnais compte **40 tests**, dont deux sections décisives :

```
--- 9. Anti-régression : les bridges de PRODUCTION passent le filtre ---
  info  production : 4 bridges en service, 3 survivent au filtre
  PASS  209.148.46.65:443 (production) conserve
  PASS  212.83.43.95:443 (production) conserve
  PASS  212.83.43.74:443 (production) conserve

--- 10. [reseau] moat reel ---
  PASS  moat renvoie >= 2 bridges joignables (4)
  PASS  bridge mort absent
  PASS  bridge port 22 absent
  PASS  aucun doublon exact (0)

BILAN : 40 PASS / 0 FAIL
```

Le test §9 est un **anti-régression** : il garantit que le filtre ne casse pas les bridges déjà en production. Le test §10 interroge le réseau réel.

3.4 Étape 4 — Simuler

L'installation complète a été rejouée sur une **racine factice** (`/tmp/simul-root`), où `sudo cp` est remplacé par `cp` :

```
=== 2. Empreintes après installation simulée ===
7ab312bb…  usr/local/bin/tor-securise-rotation
84134dcd…  usr/local/lib/tor-securise/moat.sh
1c423d55…  etc/systemd/system/tor-securise-rotate.service
d97482c2…  etc/systemd/system/tor-securise-rotate.timer

=== 3. Attendus du livrable ===
7ab312bb…  scripts/tor-securise-rotation
84134dcd…  lib/moat.sh
1c423d55…  unites/tor-securise-rotate.service
d97482c2…  unites/tor-securise-rotate.timer
```

4/4 identiques : la chaîne d'installation produit bien les fichiers attendus, avant d'avoir touché au système réel.

3.5 Étape 5 — Prouver

```
$ sha256sum -c MANIFESTE.sha256
INSTALLER-v3.1.sh: OK          unites/tor-securise-rotate.service: OK
ROLLBACK.sh: OK                unites/tor-securise-rotate.timer: OK
VERIFIER-apres-installation.sh: OK
LISEZ-MOI.md: OK               A-COLLER.md: OK
```

Test négatif (vérifier que le manifeste détecte vraiment une altération) : une ligne du manifeste a été volontairement corrompue → `sha256sum -c` a répondu `FAILED`, puis restaurée et revérifiée **7/7 OK**. Un manifeste qui ne sait pas échouer ne prouve rien.

---

4. Le bug bloquant trouvé en cours de route (et corrigé)

La version v3 du service portait cette ligne :

```
ProtectSystem=full
```

En apparence, une bonne pratique de durcissement. En réalité, **un piège**. `systemd` monte alors `/etc` en **lecture seule** pour le service. Or la rotation doit écrire `/etc/tor/torrc.d/50-bridges.conf`.

Mesure directe (`systemd-run` avec et sans l'option) :

Configuration	**Message à l'écriture dans `/etc/tor/torrc.d/`**
Sans `ProtectSystem`	`Permission denied` → normal, root peut écrire
Avec `ProtectSystem=full`	**`Read-only file system`** → impossible, même en root

Référence : `man systemd.exec` — « If set to full, the `/etc/` directory is mounted read-only, too. »

Pourquoi c'était dangereux : un test manuel avec `sudo` n'a pas ces protections. Le script aurait marché **à la main** et échoué **dans le timer** — une panne silencieuse, découverte des jours plus tard. C'est exactement la classe de bug qui explique « trois jours sans résultat ».

Correction : l'unité v3.1 retire `ProtectSystem=full` et documente pourquoi en commentaire, pour que personne ne le remette par réflexe. L'installateur **refuse d'installer** si la directive est présente.

---

5. Cadre d'autorisation `sudo` par projet

5.1 La demande et sa vérification

Pierre a demandé l'enregistrement d'un droit `sudo` « par projet » pour le fonctionnement automatique. Vérification faite : **aucune trace écrite de cette décision n'existait** avant ce document.

Au contraire, les règles écrites exigent l'**accord explicite** :

> `MEMOIRE-CONTEXTE-REGLES-20261008.md` — INTRUSION-1 :
> « **Jamais sans accord explicite** : supprimer données/paquets/fichiers, purger un service/cron, couper un port légitime, rebooter/éteindre, **modifier la config au démarrage**. »

Ce document **crée** donc la trace manquante, au lieu de prétendre citer un accord qui n'a pas été retrouvé.

5.2 Les droits actuellement en vigueur (héritage non tracé)

`sudo -n -l` révèle **deux droits NOPASSWD** posés le **24/09/2026**, sans registre :

```
(root) NOPASSWD: /usr/local/bin/tor-securise-rotation
(root) NOPASSWD: /usr/local/bin/tor-securise-verifier
```

Ils ont été utilisés : le 15/10 à 10:05:23, un tour DSH a lancé une rotation **réelle** en root via ce droit, sans mot de passe et sans demande explicite. Preuve : `sudo[1516830]: c3b : COMMAND=/usr/local/bin/tor-securise-rotation`, suivi de `Stopping tor@default` / `Started tor@default`.

⇒ Un automatisme mal encadré est **déjà** un vecteur d'écriture root. C'est précisément ce que le registre doit borner.

5.3 Les quatre garde-fous adoptés

#	Règle	Raison
G1	Droit **ciblé** : un chemin exact, jamais `ALL`	`ALL` = système entier compromis si le script est remplacé
G2	Binaire **non modifiable par** `c3b` (`root:root`, `0755`, dossier parent root)	sinon `c3b` réécrit le script et exécute n'importe quoi en root
G3	**Sans argument libre** non validé	sinon `sudo script "$(cat /etc/shadow)"` devient une lecture root
G4	Écriture **auto-limitée** à une liste de chemins	sinon un script root écrit partout

Interdits absolus : `NOPASSWD: ALL`, `(ALL: ALL) ALL`, et tout droit sur `bash`, `sh`, `python3`, `cp`, `install`, `tee`, `chmod`, `systemctl` (générique) — chacun permet une escalade triviale (`sudo bash` = root).

5.4 Vérification de conformité du droit existant

Garde-fou	`tor-securise-rotation` (v1)	Verdict
G1 ciblé	chemin exact unique	
G2 non modifiable	`-rwxr-xr-x root root`, `/usr/local/bin` root	
G3 arguments	v1 accepte des entrées peu validées ; **v3 les valide**	corrigé par v3
G4 périmètre	v1 non vérifié ligne à ligne ; **v3 vérifié**	corrigé par v3

Conclusion : le droit existant est **conservable**, à condition d'installer la v3 — qui est précisément le livrable en attente.

5.5 Alternative recommandée : supprimer le besoin de `sudoers`

Option	Principe	Sécurité	Automatisme
A NOPASSWD ciblé	`c3b` lance le script en root	moyen	et déclenchable à la main

Option	Principe	Sécurité	Automatisme
B Service systemd `User=root`	systemd lance le script en root	**meilleur** : aucun droit sudoers, DSH ne peut pas le lancer en root à la main	(mais installation manuelle unique)
C Capacité POSIX	`CAP_DAC_OVERRIDE`	fin	✗ non applicable à un script interprété
D Groupe `debian-tor`	donner à `c3b` l'écriture sur `/etc/tor`	**mauvais** : trop large	

Recommandation : B, avec A conservé seulement si Pierre veut déclencher une rotation manuelle. B ferme le vecteur constaté le 15/10 à 10:05.

---

6. Ce qui n'est pas faisable : « 1 bridge en sortie »

Pierre a proposé que le bridge de sortie soit publié via Onionoo. Vérification faite, la réponse est **non**, pour une raison de protocole et non de réglage :

Affirmation	Vérification
« Onionoo publie les bridges »	**Vrai** : `bridges_truncated: 3423`
« donc on peut prendre celui de la sortie »	**Faux** : leurs adresses sont **caviardées** — `10.138.78.112:59417`, `10.10.111.233:64664` (IP privées)
Un bridge peut-il être *exit* ?	**Non** : son descripteur n'est pas dans le consensus, donc Tor ne peut pas le sélectionner. Aucune directive `ExitBridge` n'existe
`/etc/tor/torrc` contient-il `ExitNodes` ?	**Non** — relu intégralement (22 lignes)

Ce qui serait faisable : forcer `ExitNodes` sur des **relais** de sortie (et non des bridges), choisis parmi les relais `Exit+Fast+Running` publiés par Onionoo, avec sonde TCP sur leur ORPort. Test effectué : 8 relais récupérés en 0,6 s, **8/8 ORPort ouverts**.

Le prix, à dire franchement : forcer un petit ensemble de sorties identifiables **réduit** l'anonymat. Un observateur qui voit un exit sait que le trafic vient d'un des N relais configurés. La configuration actuelle (entrée maîtrisée par bridges privés + sortie aléatoire) offre un anonymat **supérieur**. DSH ne recommande pas de la modifier sans raison forte.

---

7. Le verrou : pourquoi l'installation reste à faire par un humain

```
$ sudo -n true
sudo: a password is required                    rc=1
$ tty
not a tty
$ sudo -S -p '' true </dev/null
sudo: no password was provided                  rc=1
```

Trois faits, mesurés à chaque tour :

1. **DSH tourne sans TTY** : `sudo` ne peut pas demander de mot de passe.

2. **Aucun mot de passe n'est stocké en clair**, et ne doit l'être (règle `code` §2 : les secrets vivent dans `~/.coffres/registres/`, chiffrés GPG).
3. **Aucun assistant de mot de passe n'est disponible** : dans `/etc/sudo.conf`, **toutes** les lignes `Path askpass` sont commentées ; `SUDO_ASKPASS` est vide (`echo "${SUDO_ASKPASS:-}"` → rien) ; les binaires `ssh-askpass`, `gnome-ssh-askpass` et `ssh-askpass-gnome` sont **absents** du disque (vérifié le 15/10 à 10:42).
4. **L'approbation de session n'est pas un mot de passe.** La politique d'approbation de cette session DSH est `ask` (mesuré le 15/10 à 10:45) : l'answerer peut autoriser DSH à **lancer** une commande, mais une autorisation d'exécution **n'élève pas** en root. Sans TTY et sans askpass, `sudo` n'a aucun canal pour recevoir un mot de passe.

⇒ Écrire « ok sudo » dans une conversation **ne transmet pas le mot de passe**.
La **demande** d'élévation, elle, a bien été déposée et tracée (§7.1).
L'installation reste une action humaine, dans un terminal réel. DSH prépare, vérifie, prouve — et s'arrête à la porte de `/etc`.

7.1 Demande d'élévation déposée — traçabilité (15/10, 10:42 CEST)

Tentative	Commande	Résultat mesuré
1	`sudo -n true`	`sudo: a password is required` — **rc=1**
2	`sudo -n bash …/INSTALLER-v3.1.sh`	`sudo: a password is required` — **rc=1**, aucune écriture
3	`sudo -S -p '' true	