RomM — Forcer un core EmulatorJS par plateforme

> *Dernière mise à jour : 2026-09-06*

EmulatorJS propose souvent **plusieurs cores** (moteurs d'émulation) pour une même plateforme, et ils ne se valent pas avec notre bibliothèque. RomM laisse normalement le choix à l'utilisateur et mémorise sa décision **dans son navigateur** — ce qui va mal avec une instance partagée entre plusieurs personnes et plusieurs appareils.

Cette page décrit le patch qui impose un core côté serveur, pour tout le monde :

```
Nintendo 64 → parallel_n64
Arcade      → mame2003_plus
```

Toutes les autres plateformes gardent le comportement normal de RomM.

> ⚠ **Il y a deux patchs EmulatorJS sur cette instance, et ils ne touchent pas le même fichier.** Les noms se ressemblent beaucoup, d'où une confusion facile :
>
> | Patch | Fichier modifié | Ce que c'est | Persistance |
> |---|---|---|---|
> | **Manettes** (page « RomM — Manettes dans EmulatorJS ») | `assets/emulatorjs/data/src/emulator.js` | le **moteur** EmulatorJS lui-même | bind mount Unraid |
> | **Cores** (cette page) | `assets/EmulatorJS-*.js` | le **bundle frontend de RomM** qui pilote le moteur | script relancé par *Post Arguments* |
>
> Ils sont indépendants et coexistent sans conflit. Le `emulatorjs: debug: true` du `config.yml`, exigé par le patch manettes, **n'a aucun effet** sur celui-ci.

>

> 🖥️ **Contexte** : RomM **5.2.0**, image `rommapp/romm:latest`, conteneur nommé `romm`, sur Unraid.

---

1. Le problème — un choix qui appartient au navigateur

RomM mémorise le core retenu dans le `localStorage` du navigateur, sous deux clés :

```
player:<ROM_ID>:core          → le core choisi pour un jeu précis
player:<PLATFORM_SLUG>:core   → le core choisi pour toute une plateforme
```

Par exemple `player:n64:core` ou `player:arcade:core`.

Trois conséquences gênantes :

- la préférence ne suit ni l'utilisateur ni l'appareil — chaque navigateur repart de zéro ;
- un utilisateur peut sélectionner un core par mégarde, obtenir un jeu qui rame ou ne démarre pas, et **conclure que le jeu est cassé** alors que seul le core est en cause ;
- rien ne permet à l'administrateur de dire « sur cette plateforme, c'est ce core-là ».

2. Comment RomM choisit un core

Dans RomM 5.2.0, l'ordre de sélection est :

```
1. Core mémorisé pour le jeu        (player:<ROM_ID>:core)
2. Core mémorisé pour la plateforme (player:<PLATFORM_SLUG>:core)
3. Premier core supporté par RomM
```

Le premier de la liste qui est à la fois renseigné **et** présent dans les cores supportés l'emporte.

3. Pourquoi un patch, et pas un réglage

RomM ne propose aujourd'hui **aucun paramètre serveur** pour imposer un core. Ce genre de configuration n'existe pas :

```
n64:
  default_core: parallel_n64
```

La section `emulatorjs.settings` du `config.yml` permet de régler les **options** d'un core, mais pas de désigner celui qui doit être choisi. Il faut donc modifier la logique du frontend.

4. Ce que fait le patch

Un core « administrateur » est inséré **en tête** de la liste de candidats :

```
1. Core forcé par l'administrateur   ← ajouté par le patch
2. Core mémorisé pour le jeu
3. Core mémorisé pour la plateforme
4. Premier core supporté
```

Concrètement, le patch injecte une table de correspondance dans le bundle :

```
{n64:`parallel_n64`, arcade:`mame2003_plus`}[rom.value.platform_slug]
```

Deux garde-fous en découlent naturellement :

- **Plateforme absente de la table** → la valeur est `undefined`, elle est ignorée, et RomM retrouve son comportement normal. C'est le cas de toutes les plateformes autres que N64 et Arcade.
- **Core forcé absent des cores supportés** → il est également écarté, car la valeur injectée traverse le même filtre que les autres (`cores.includes(...)`). Si une future version de RomM retirait `parallel_n64`, le lancement ne se bloquerait pas : RomM reprendrait simplement sa sélection habituelle.

> Les préférences déjà enregistrées dans les navigateurs **ne sont pas effacées**. Elles restent en place et redeviendraient actives si le patch était retiré ; elles sont simplement ignorées pour les deux plateformes forcées, puisque le core administrateur passe avant.

---

5. Le script

5.1 Le dossier

```
mkdir -p /mnt/user/appdata/romm-core-patch
```

5.2 Créer le script

```
cat > /mnt/user/appdata/romm-core-patch/apply.sh <<'EOF'
#!/bin/bash
CONTAINER="romm"

# Attend que le conteneur RomM soit démarré
for i in $(seq 1 60); do
    if docker inspect -f '{{.State.Running}}' "$CONTAINER" 2>/dev/null | grep -q true; then
        break
    fi
    sleep 2
done

docker exec -i "$CONTAINER" python3 - <<'PY'
from pathlib import Path
import re
import sys

assets = Path("/var/www/html/assets")
files = list(assets.glob("EmulatorJS-*.js"))
marker = "/*ROMM_FORCE_CORE*/"

pattern = re.compile(
    r'(?P<sel>[A-Za-z_$][\w$]*)\.value=\['
    r'localStorage\.getItem\(`player:\$\{(?P<rom>[A-Za-z_$][\w$]*)\.value\.id\}:core`\),'
    r'localStorage\.getItem\(`player:\$\{(?P=rom)\.value\.platform_slug\}:core`\)'
    r'\]\.find\((?P<x>[A-Za-z_$][\w$]*)=>!!(?P=x)&&'
    r'(?P<cores>[A-Za-z_$][\w$]*)\.value\.includes\((?P=x)\)\)'
    r'\?\?(?P=cores)\.value\[0\]'
```

```
)

matches = []
for p in files:
    text = p.read_text(errors="replace")
    if marker in text:
        print(f"[OK] Patch déjà présent : {p.name}")
        sys.exit(0)
    m = pattern.search(text)
    if m:
        matches.append((p, text, m))

if len(matches) != 1:
    print(f"[ERREUR] {len(matches)} bundle(s) compatible(s) trouvé(s).")
    print("Aucune modification effectuée.")
    sys.exit(1)

p, text, m = matches[0]
sel = m.group("sel")
rom = m.group("rom")
old = m.group(0)
prefix = f"{sel}.value=["

forced = (
    f'{prefix}{marker}'
    f'{{n64:`parallel_n64`,arcade:`mame2003_plus`}}'
    f'[{rom}.value.platform_slug],'
)
new = old.replace(prefix, forced, 1)

backup = Path(str(p) + ".before-core-patch")
if not backup.exists():
    backup.write_text(text)

text = text[:m.start()] + new + text[m.end():]
p.write_text(text)

print(f"[OK] Patch appliqué : {p.name}")
print("[OK] N64    -> parallel_n64")
```

```
print("[OK] Arcade -> mame2003_plus")
PY
EOF
```

5.3 Le rendre exécutable

```
chmod +x /mnt/user/appdata/romm-core-patch/apply.sh
```

> ⚠ Sans ce `chmod`, le lancement par *Post Arguments* (§7) échoue en silence à chaque recréation du conteneur.

5.4 Ce que fait le script, étape par étape

1. attend que le conteneur `romm` soit actif (60 tentatives, 2 secondes d'intervalle — soit 2 minutes au plus) ;
2. cherche le bundle `/var/www/html/assets/EmulatorJS-*.js`, dont le nom est **hashé et change à chaque version** ;
3. s'arrête immédiatement si le marqueur `/*ROMM_FORCE_CORE*/` est déjà présent ;
4. localise la logique de sélection du core par expression régulière ;
5. **exige exactement une correspondance** — ni zéro, ni plusieurs — sinon il ne touche à rien ;
6. sauvegarde le bundle d'origine en `.before-core-patch` ;
7. insère la table de correspondance et écrit le fichier.

Le script est donc **idempotent** : le relancer dix fois donne le même résultat qu'une fois.

---

6. Appliquer et vérifier

6.1 Lancement manuel

```
/mnt/user/appdata/romm-core-patch/apply.sh
```

Résultat attendu à la première exécution :

```
[OK] Patch appliqué : EmulatorJS-XXXXXXXX.js
[OK] N64    -> parallel_n64
[OK] Arcade -> mame2003_plus
```

Et aux suivantes :

```
[OK] Patch déjà présent : EmulatorJS-XXXXXXXX.js
```

6.2 Vérifier que le patch est en place

```
docker exec romm sh -c '
grep -RlF "/*ROMM_FORCE_CORE*/" /var/www/html/assets/EmulatorJS-*.js
'
```

Le chemin du bundle doit s'afficher :

```
/var/www/html/assets/EmulatorJS-XXXXXXXX.js
```

Une réponse vide signifie que le patch n'est pas appliqué.

6.3 ⚠ Vider le cache du navigateur

Le nom du bundle est hashé, ce qui pousse les navigateurs à le mettre en cache **très longtemps**. Or le patch change son **contenu sans changer son nom** : un navigateur qui l'a déjà chargé continuera de servir la version non patchée, parfois plusieurs jours.

Après chaque application du patch, forcer le rechargement :

```
Windows / Linux : Ctrl + Shift + R
macOS           : Cmd + Shift + R
```

C'est la première chose à vérifier quand le patch est présent côté serveur mais que le core ne change pas côté joueur — et il faut le faire **sur chaque appareil** utilisé.

---

7. Persistance — pourquoi un script et pas un bind mount

Le patch manettes est rendu persistant par un montage de fichier. **Cette méthode ne convient pas ici**, pour une raison simple : le nom du bundle est hashé et change à chaque mise à jour.

```
EmulatorJS-Dr7gVfBj.js   →   EmulatorJS-AbCd1234.js
```

Monter un fichier sur un nom hashé reviendrait à figer une version qui n'existera plus, et à masquer le vrai bundle de la nouvelle image. C'est pourquoi on **repatche** au lieu de **remplacer**.

> Cette approche a un avantage que le bind mount n'a pas : après une mise à jour de RomM, c'est bien le **nouveau** fichier qui est patché. Il n'y a aucun risque de servir un vieux fichier à un runtime récent.

La solution : Post Arguments

Dans le template Docker RomM d'Unraid, passer en **Advanced View**, puis ajouter dans le champ **Post Arguments** :

```
&& /mnt/user/appdata/romm-core-patch/apply.sh
```

Unraid construit alors une commande de la forme :

```
docker run ... rommapp/romm:latest && /mnt/user/appdata/romm-core-patch/apply.sh
```

Le `docker run` rend la main dès que le conteneur démarre — d'où la boucle d'attente du script, qui laisse à RomM le temps d'être réellement actif avant de le patcher.

8. Ce qui se passe dans chaque situation

Action	Conteneur recréé ?	Bundle réinitialisé ?	Patch après l'opération
`docker restart romm`	non	non	**toujours là** — rien à faire
Mise à jour de l'image RomM	oui	oui	réappliqué automatiquement
`Apply` du template Unraid	oui	oui	réappliqué automatiquement

Action	Conteneur recréé ?	Bundle réinitialisé ?	Patch après l'opération
Modification du mapping (§10)	non	non	à relancer à la main

Aucune intervention manuelle n'est donc nécessaire après une mise à jour de RomM — **à condition que le patch s'applique toujours**, ce qui est le sujet de la section suivante.

---

9. ⚠ Après une mise à jour de RomM

L'expression régulière du script vise du JavaScript **minifié et très précis**. Il suffit que RomM réécrive cette portion de code, ou que le minificateur produise autre chose, pour qu'elle ne corresponde plus.

Dans ce cas le script affiche :

```
[ERREUR] 0 bundle(s) compatible(s) trouvé(s).
Aucune modification effectuée.
```

Le comportement est volontairement prudent : il ne casse rien. RomM continue de fonctionner normalement, avec sa sélection de core habituelle.

> ⚠ **Mais l'échec est silencieux pour l'utilisateur.** Lancé par *Post Arguments*, le script écrit son message d'erreur là où personne ne le lit. Personne n'est prévenu : les cores cessent simplement d'être forcés, et le symptôme réapparaît sous la forme « le jeu N64 ne marche plus » plusieurs jours plus tard.
>
> **Après toute mise à jour de RomM, lancer la vérification du §6.2.** Une réponse vide veut dire que le patch n'est plus appliqué.

Pour comprendre pourquoi, relancer le script à la main (§6.1) et lire son message. Si l'expression régulière ne correspond plus, il faut la réécrire d'après le nouveau code du bundle — le fichier `.before-core-patch` de l'ancienne version, s'il est encore là, aide à repérer ce qui a changé.

> Le fichier `.before-core-patch` est créé **dans le conteneur**. Il disparaît à chaque recréation : ce n'est pas une sauvegarde durable, seulement un filet pour la

> session en cours. Ce n'est pas gênant — l'original revient toujours de l'image Docker.

---

10. Ajouter une plateforme

Le mapping se trouve dans cette ligne du script :

```
f'{{n64:`parallel_n64`,arcade:`mame2003_plus`}}'
```

> ⚠ **Les accolades doubles sont obligatoires.** Cette ligne est une *f-string* Python : `{{` et `}}` y produisent une simple accolade dans le JavaScript final. En écrivant `{n64:...}` avec des accolades simples, Python interpréterait le contenu comme une variable et le script échouerait.

Pour ajouter une plateforme, allonger la liste **à l'intérieur** des accolades doubles :

```
f'{{n64:`parallel_n64`,arcade:`mame2003_plus`,gba:`mgba`}}'
```

Deux valeurs doivent être exactes, sans quoi la plateforme sera simplement ignorée :

- le **slug RomM** de la plateforme (celui qui apparaît dans l'URL de la plateforme, et dans la clé `player:<slug>:core`) ;
- le **nom du core EmulatorJS**, tel qu'EmulatorJS le désigne.

Après modification, relancer le script — il ne fera rien tant que l'ancien patch est en place, il faut donc **d'abord recréer le conteneur** (ou restaurer le bundle) pour repartir d'un fichier propre.

---

11. Revenir au comportement standard

Retirer des **Post Arguments** du template :

```
&& /mnt/user/appdata/romm-core-patch/apply.sh
```

puis faire **Apply**. Unraid recrée RomM depuis l'image officielle, et le bundle repart non patché.

Un simple `docker restart` ne suffit pas : le conteneur doit être **recréé** pour que le bundle d'origine revienne.

Le dossier `/mnt/user/appdata/romm-core-patch/` peut être conservé — il ne gêne plus, et garde le script sous la main.

> Les préférences de core enregistrées dans les navigateurs redeviennent alors actives. Un utilisateur qui avait choisi un core inadapté avant le patch le retrouvera.

---

12. Dépannage rapide

Symptôme	Cause probable	Solution
Le core n'est pas forcé, patch pourtant présent côté serveur	Cache du navigateur : ancien bundle servi	Rechargement forcé, sur chaque appareil — §6.3
`grep` du §6.2 ne renvoie rien après une mise à jour	L'expression régulière ne correspond plus au nouveau code	Relancer le script à la main et lire l'erreur — §9
`[ERREUR] 0 bundle(s) compatible(s) trouvé(s).`	RomM a réécrit la logique de sélection du core	Réécrire l'expression régulière d'après le nouveau bundle — §9
`[ERREUR] 2 bundle(s) compatible(s) trouvé(s).`	Plusieurs bundles présents dans `assets/`	Vérifier le contenu du dossier ; le script refuse volontairement de deviner
Rien ne se passe après une recréation du conteneur	Script non exécutable, ou *Post Arguments* absent	`chmod +x` (§5.3) et vérifier le template (§7)
Le patch disparaît après un `Apply` du template	*Post Arguments* effacé lors d'une édition du template	Le remettre — §7
Une nouvelle plateforme n'est pas forcée	Slug ou nom de core erroné, ou accolades simples	§10
Le core forcé n'est pas pris alors que le patch est là	Ce core n'est plus dans les cores supportés par RomM	Comportement voulu — vérifier le nom du core, §4
Le conteneur a été renommé, le script ne fait plus rien	`CONTAINER="romm"` est écrit en dur dans le script	Corriger la variable en tête du script — §13

---

13. À compléter

- **Le nom du conteneur est en dur** (`CONTAINER="romm"`), alors que les autres pages RomM le détectent par son image : `ROMM="$(docker ps --format '{{.Names}} {{.Image}}' | awk '$2 ~ /rommapp\/romm/ {print $1; exit}')"`. Aligner le script sur cette méthode le rendrait insensible à un renommage.
- **Aucune trace n'est conservée des exécutions.** Rediriger la sortie vers un fichier (`>> /mnt/user/appdata/romm-core-patch/apply.log 2>&1` dans les *Post Arguments*) permettrait de savoir, après coup, si le patch a réussi lors de la dernière mise à jour — c'est le point faible identifié au §9.
- **Pourquoi ces cores plutôt que d'autres ?** Noter ici ce qui a été constaté avec les autres cores N64 et Arcade : cela évitera de refaire les essais, et aidera à choisir un remplaçant si l'un d'eux disparaît.
- **Demande en amont ?** Un réglage serveur de core par défaut serait la vraie solution. Vérifier si la fonctionnalité est déjà demandée au projet RomM, et suivre le ticket le cas échéant.

Revision #3
Created 2026-09-06 12:18:00 UTC by thymon
Updated 2026-09-06 12:28:34 UTC by thymon