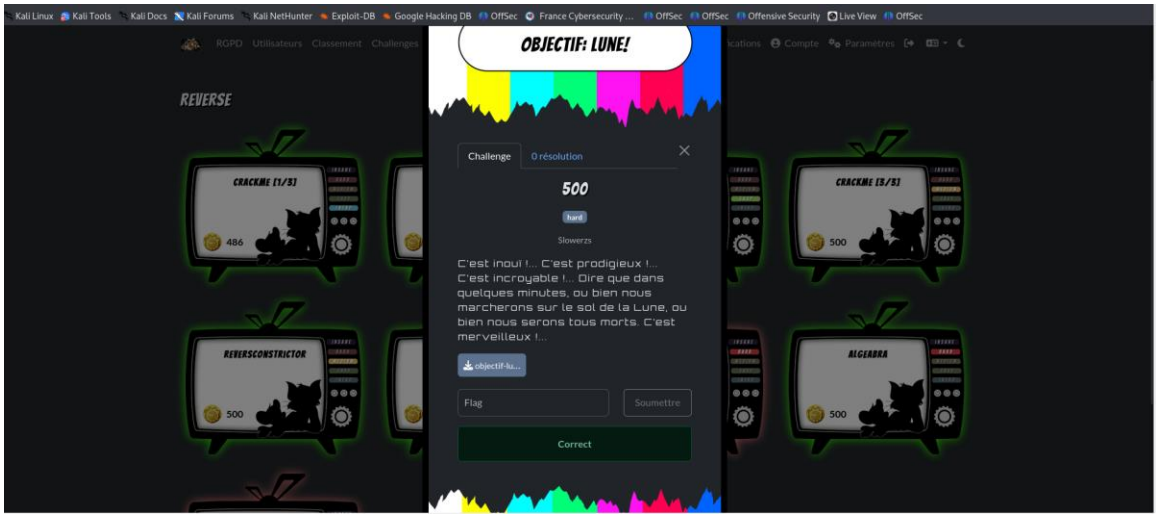
# Start.hackademint — Objectif Lune

Catégorie : Reverse • Format : Mach-O arm64 • Objet : déchiffrement de flag

Writeup

## Rapport de résolution




## Résumé exécutif

Le challenge fournit un exécutable Mach-O (arm64) et un fichier de flag chiffré. Le binaire embarque un couple clair/chiffré de référence (buffers de 256 octets) utilisé pour un auto-contrôle. En supposant un chiffrement de type flux (XOR avec un keystream), ce couple permet de reconstruire le keystream par XOR puis de déchiffrer directement le fichier flag.

## 1. Données d'entrée et outillage

L'objectif de cette section est de fixer le périmètre (fichiers fournis), l'environnement de travail et les commandes minimales permettant de reproduire l'extraction des données et le déchiffrement.

### 1.1 Fichiers fournis

• objectif_lune : exécutable Mach-O 64 bits pour arm64 (Apple Silicon). Il contient la logique applicative et, surtout, des constantes utilisées pour un auto-contrôle cryptographique.

• flag.txt.enc : fichier chiffré contenant le flag (71 octets dans l'instance analysée).

### 1.2 Hypothèse de travail et indicateurs observables

Le challenge oriente vers une résolution « reverse et une crypto légère » : on ne cherche pas à casser une primitive moderne, mais à exploiter une erreur de conception (auto-test clair/chiffré embarqué). Les indicateurs qui guident cette hypothèse sont :

• présence d'un self-check (données « expected ») dans le binaire ;
• taille courte et non multiple d'un bloc (71 octets), compatible avec un schéma de type flux (XOR/keystream) ;

• absence d'IV/nonce explicite livré avec le chiffré, ce qui rend plausible une réinitialisation deterministe du flot au démarrage du chiffrement.

## 1.3 Environnement et dépendances

• OS : Kali Linux (ou équivalent).
• Python : 3.8+ (recommandé 3.10+). Par ici 3.13 et au-delà.
• Aucun module externe requis : le solveur utilise uniquement la bibliothèque standard (struct, argparse, pathlib).
• Outils optionnels (diagnostic) : file, strings, otool/llvm-objdump, Ghidra/Cutter/radare2.

## 1.4 Commandes de reproduction

1) Vérifier le type de binaire :
file objectif_lune
2) Rechercher des indices textuels (si symboles non strippés) :
strings -a objectif_lune | grep -i expected
3) Exécuter le solveur en mode preuve (recommandé) :
python3 decrypt_objectif_lune.py -v
4) Écrire le clair dans un fichier (optionnel) :
python3 decrypt_objectif_lune.py -v --out flag.txt

## 1.5 Ce que le mode -v doit démontrer

Le mode verbeux doit permettre à un lecteur tiers de valider, sans interprétation, les points suivants :
• sélection de slice arm64 si binaire FAT/universal ;
• endianness et cohérence des load commands (ncmds, sizeofcmds, cmdsize bornés) ;
• sections Mach-O pertinentes (__TEXT, __DATA, __LINKEDIT) et offsets fichier ;
• stratégie de récupération des buffers de self-check : via symboles si adresses valides, sinon fallback par extraction de 512 octets dans __DATA,__data ;
• test des deux ordres possibles des buffers 2×256 et validation par un critère de plausibilité (préfixe Star{…}).

## 1.6 Résultat attendu

À l'issue de l'exécution, le solveur doit afficher un flag au format Star{...} (et idéalement permettre l'écriture du clair via --out).

Outil maison : de résolution :

- decrypt_objectif_lune.py (solveur Python)

Outils recommandés (analyse) :

- Ghidra (ou Cutter/radare2) pour l'inspection statique
- otool / llvm-objdump (optionnel)
- Python 3 pour exécuter le solveur

## 2. Analyse technique

### 2.1 Format Mach-O et implications

Le binaire est un Mach-O arm64 (éventuellement FAT/universal). La résolution nécessite un parsing fiable des load commands (LC_SEGMENT_64, LC_SYMTAB) et une gestion robuste de l'endianness. Les erreurs classiques proviennent d'un mauvais cmdsize entraînant des offsets hors bornes.

### 2.2 Hypothèse cryptographique exploitée

Le schéma exploité est celui d'un chiffrement par XOR avec un flot (keystream) : C = P XOR KS. Dès qu'un couple (P_test, C_test) est connu, on en déduit KS = P_test XOR C_test, puis on déchiffre le flag par P_flag = C_flag XOR KS (sur la longueur utile).

### 2.3 Localisation des buffers attendus

Deux approches sont possibles : (1) résolution par symboles (_expected_data, _expected_enc_data) via LC_SYMTAB, si leurs adresses sont exploitables ; (2) fallback par section : lorsque les symboles existent mais ont addr=0, on extrait le payload dans la section __DATA,__data, typiquement de taille 0x200 (512) = 2 × 256 octets.

## 3. Déroulement de résolution (≤ 10 étapes)

1. Ouvrir le binaire et confirmer le format Mach-O arm64 (optionnel : `file objectif_lune`).
2. Lancer le solveur en mode verbeux : `python3 decrypt_objectif_lune.py -v`.
3. Sélectionner automatiquement la slice arm64 si le binaire est FAT.
4. Valider l'endianness par cohérence header + bornes des load commands.
5. Lister les sections (__TEXT, __DATA, __LINKEDIT) et leurs offsets fichier.
6. Tenter la résolution des symboles `_expected_data` / `_expected_enc_data` via LC_SYMTAB.
7. Si les symboles sont non résolus (addr=0), extraire 512 octets depuis `__DATA,__data` et les découper en deux buffers de 256 octets.
8. Calculer le keystream par XOR des deux buffers (tester les deux ordres possibles).
9. Déchiffrer `flag.txt.enc` par XOR avec le keystream et sélectionner la sortie plausiblement formatée `Star{...}`.
10. Afficher le flag et, si besoin, l'écrire en sortie via `--out`.

## 4. Reproduction (commandes)

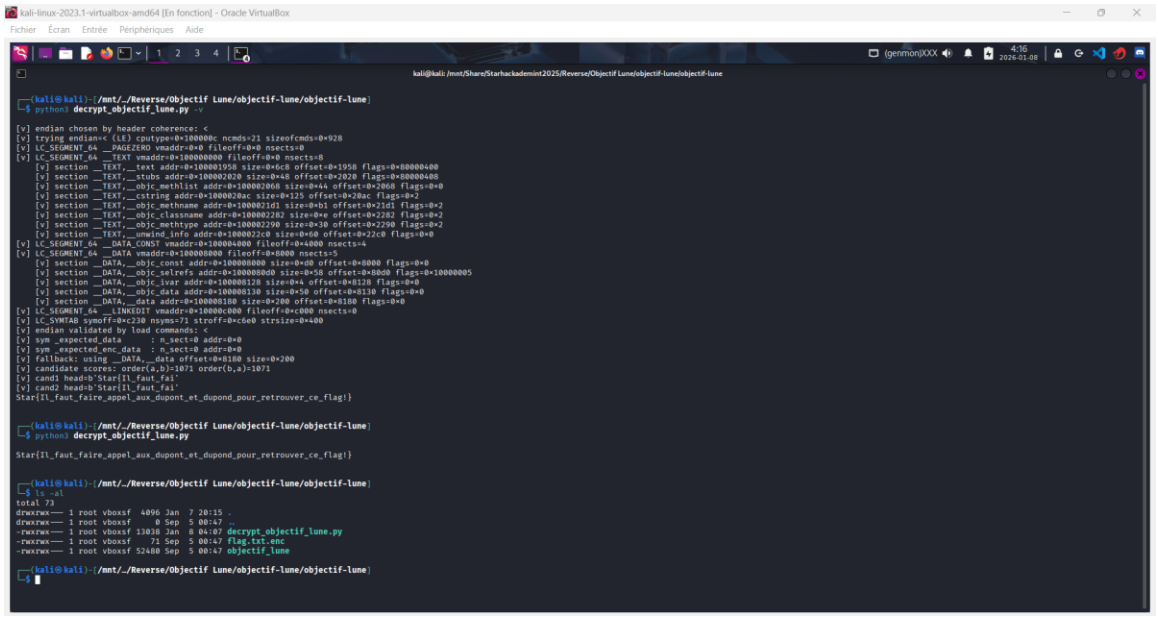
**Mode standard :**

python3 decrypt_objectif_lune.py

**Mode verbeux (preuve) :**

python3 decrypt_objectif_lune.py -v

**Écriture du clair :**
python3 decrypt_objectif_lune.py --out flag.txt




## 5. Résultat

Flag récupéré :

*Star{Il_faut_faire_appel_aux_dupont_et_dupond_pour_retrouver_ce_flag!}*

## 6. Bugs résolus et pièges à éviter

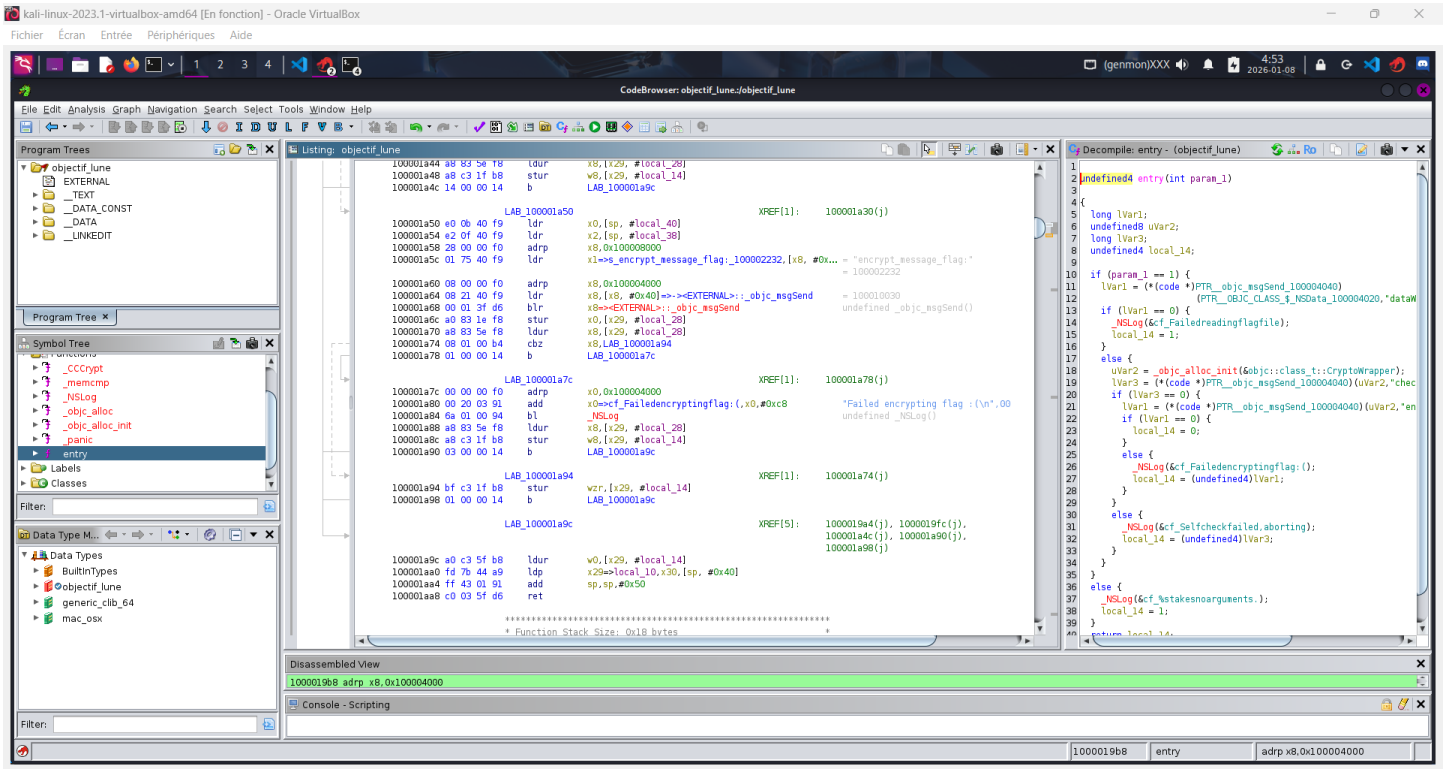
Ce challenge Reverse combine un binaire Mach-O arm64 et un fichier chiffré. La difficulté ne réside pas dans une crypto sophistiquée, mais dans les détails de parsing Mach-O et dans l'identification correcte des buffers de self-check qui permettent de reconstruire un keystream.

### 6.1 Principaux bugs résolus (côté solveur)

- Détection d'endianness Mach-O : confusion entre MH_MAGIC_64 et MH_CIGAM_64 (byteswapped) menant à des offsets absurdes (cmdsize/ncmds incohérents) et à des lectures hors bornes.
- Gestion des binaires FAT/universal : nécessité de sélectionner explicitement la slice arm64 avant tout parsing Mach-O.
- Validation stricte des bornes : ajout de garde-fous sur sizeofcmds, cmdsize et la plage réelle des load commands afin d'éviter les accès à des offsets > taille du fichier.
- Mapping adresse → section : correction de la logique de localisation des données (ne pas se fier aveuglément à n_sect ; préférer la recherche par inclusion d'intervalle [addr, addr+size)).
- Symboles "inutilisables" : `_expected_data` et `_expected_enc_data` présents en SYMTAB mais avec addr=0x0 / n_sect=0 (symboles indéfinis/strip partiel). Passage à un fallback basé sur la section __DATA,__data.
- Bug de verbose : protection contre les cas `sec_p=None` / `sec_e=None` pour éviter les TypeError et produire un diagnostic exploitable.

### 6.2 Pièges classiques à éviter

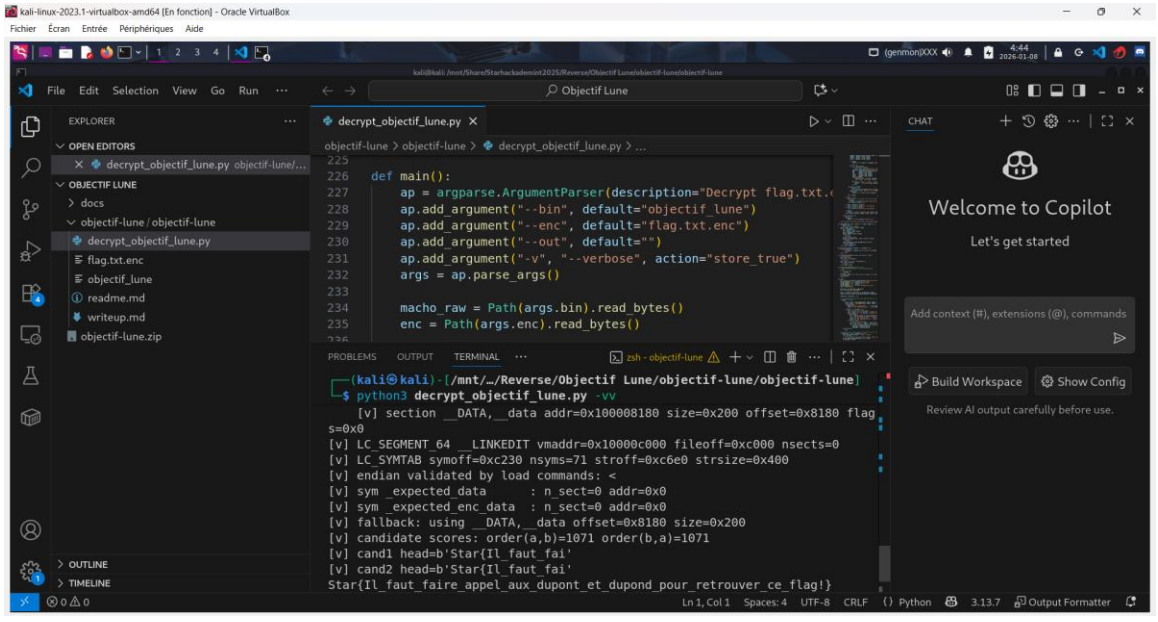
- Se focaliser sur `key.bin` : le programme évoque une clé externe, mais la résolution exploite surtout le couple (clair/chiffré) du self-check intégré au binaire.
- Supposer AES-CBC/PKCS7 "par défaut" : la taille de sortie et l'absence d'IV stocké orientent vers un schéma de flux (XOR avec keystream). Toujours partir d'observables (longueurs, présence d'IV/nonce, appels API).
- Croire que la table des symboles suffit : sur Mach-O, les noms peuvent exister sans valeur exploitable (addr=0). Il faut prévoir une stratégie de repli (analyse de sections, pattern, désassemblage).

- Confondre adresses virtuelles et offsets fichier : le calcul correct est `file_offset = section.offset + (sym_addr - section.addr)` — et uniquement si la section est file-backed.
- Ne pas instrumenter : le mode `-v` n'est pas cosmétique ; il doit prouver endianness, load commands, sections, offsets, et la lecture effective des 512 octets (2×256) permettant le keystream.
- Oublier de tester l'ordre des buffers : si la section contient 512 octets, il peut s'agir de `P||C` ou `C||P`. Tester les deux et valider via un critère (préfixe `Star{`, ratio imprimable).
- 

## 6.3 Lecture "preuve" en une équation

Si le chiffrement est de type flux : C = P ⊕ KS. Le self-check fournit (P_test, C_test) ⇒ KS = P_test ⊕ C_test. Puis flag = C_flag ⊕ KS[0:len(C_flag)].

# Annexes




***Capture de de l'écran de lancement du script conduisant au résultat :le drapeau (flag)***