Blockchain Solana Security (Proof of Concept)

Rapport Technique :

Auteur : VERGEROLLE Loicq

📋 Table des Matières

💻 Introduction et Contexte

1.1 Définition et Portée du Projet

Ce projet de recherche en sécurité blockchain propose une analyse approfondie et pratique de deux vecteurs d'attaque sophistiqués permettant d'exécuter des **rugpulls** sur l'écosystème Solana. Notre étude se concentre spécifiquement sur :

1. **L'exploit d'Authority Update** : Une vulnérabilité critique permettant à un attaquant de subtiliser les fonds verrouillés dans un smart contract via une backdoor contractuelle ingénieuse. Contrairement aux failles conventionnelles répertoriées (CVE), cette vulnérabilité repose sur une mauvaise implémentation des mécanismes de permission des programmes Solana, exploitant notamment les faiblesses dans la gestion des PDA (Program Derived Addresses) et des autorités de mise à jour. L'absence de CVE publique rend cette faille particulièrement insidieuse, car elle échappe aux outils de détection automatisés et aux bases de données de vulnérabilités connues.

2. **Le mécanisme de manipulation de marché** : Une attaque combinant des techniques avancées de fragmentation de transactions (bundles) et l'utilisation

d'un bot de trading algorithmique conçu pour simuler artificiellement l'activité d'investisseurs réels. Ce système crée une illusion de demande organique, provoquant une inflation artificielle du prix du token avant que l'attaquant ne liquide brusquement ses positions. L'approche utilise des bundles fragmentés pour contourner les détections de front-running et des modèles de trading génératifs (via du machine learning simple) pour reproduire des comportements d'achat crédibles.

L'objectif principal de ce projet est triple :

- **Exposer des vulnérabilités structurelles** dans la conception des smart contracts Solana, souvent négligées lors des audits traditionnels qui se concentrent sur les bugs évidents plutôt que sur les failles architecturales.
- **Démontrer opérationnellement** comment ces attaques peuvent être orchestrées, en fournissant des preuves de concept (PoC) fonctionnelles permettant à la fois de reproduire les exploits et de développer des contre-mesures.
- **Alimenter la recherche en sécurité blockchain** en documentant des vecteurs d'attaque émergents qui combinent des failles techniques avec des manipulations psychologiques (social engineering) pour maximiser leur impact.

Cette étude s'adresse principalement aux :

- **Développeurs DeFi** pour les aider à sécuriser leurs contrats contre ces attaques avancées
- **Auditeurs de smart contracts** afin d'enrichir leurs méthodologies d'analyse
- **Chercheurs en sécurité blockchain** intéressés par l'évolution des techniques d'exploitation
- **Investisseurs avertis** souhaitant comprendre les risques sous-jacents aux projets DeFi

La méthodologie combine :

- Une analyse statique et dynamique du code des smart contracts
- Des tests en environnement sandbox (testnet/local validator)
- Une modélisation des attaques avec des scénarios réalistes
- Le développement d'outils de détection personnalisés

Contrairement aux analyses théoriques, ce projet fournit des implémentations concrètes permettant de :

- Reproduire l'exploit d'Authority Update via un contrat Rust minimaliste

Les implications de cette recherche sont significatives pour l'écosystème Solana, révélant comment :

- Les permissions mal configurées peuvent transformer une fonctionnalité légitime en backdoor
- La vitesse et le faible coût des transactions facilitent les manipulations de marché
- Les outils d'audit actuels présentent des angles morts dangereux

En documentant ces attaques de manière exhaustive, je vive à établir de nouvelles best practices pour :

- La sécurisation des mécanismes d'autorité dans les programmes Solana
- La détection des tentatives de manipulation de liquidités
- L'éducation des participants à l'écosystème DeFi

Ce projet s'inscrit dans une démarche de sécurité offensive constructive, où la compréhension approfondie des vecteurs d'attaque permet de bâtir des défenses plus robustes pour tout l'écosystème blockchain.

A. Exploit d'Authority Update (Backdoor Silencieuse)

Fonctionnement Technique Approfondi

Cet exploit cible spécifiquement les smart contracts Solana implémentant des mécanismes de mise à jour (*upgradeable authorities*) sans verrouillage sécurisé (*freeze_authority*). Il repose sur une faille de conception critique dans la gestion des permissions, permettant à un attaquant de subtiliser les fonds de manière furtive.

Mécanisme d'Attaque Détaillé

1. **Cible Privilégiée** :
   Les contrats utilisant des PDA (*Program Derived Addresses*) comme autorité modifiable, sans implémenter de timelock ou de multi-signature pour les changements sensibles.
   Particulièrement vulnérables : les pools de liquidité, les contrats de staking, et les mint authorities de tokens.

2. **Technique d'Exploitation** :

   Injection d'Instruction Cachée :
   L'attaquant intègre discrètement dans le program_id une logique permettant de modifier l'authority via une instruction non documentée (ex : update_hidden_authority).

   Contournement des Vérifications :
   Certaines librairies tierces (ex : versions obsolètes d'Anchor) ne valident pas

rigoureusement les `signers des CPI (*Cross-Program Invocations*), autorisant des appels malveillants.

Absence de CVE :
Contrairement aux bugs classiques, cette vulnérabilité résulte d'une *mauvaise pratique de développement* (ex : omission de require_auth), non répertoriée dans les bases de vulnérabilités publiques.

3. **Exécution de l'Exploit** :

   Phase de Dormance : Le contrat fonctionne normalement jusqu'au déclenchement.
   Activation : L'attaquant appelle la fonction cachée pour :

   - Réattribuer l'authority à un wallet sous son contrôle.
   - Drainer les fonds via un transfert légitime en apparence (ex : token::transfer).

Impact et Furtivité

- **Drainage Instantané** :
  Tous les actifs du contrat (tokens, SOL) peuvent être siphonnés en une seule transaction.
- **Traces Minimales** :
  Aucune modification visible du bytecode du contrat (contrairement aux upgrades malveillantes).
  Les logs on-chain montrent simplement un transfert "autorisé" par l'authority.
- **Détection Complexe** :
  Les outils d'audit statique (ex : Slither) passent à côté car :
  - La backdoor est noyée dans des centaines de lignes de code.
  - L'exploit n'utilise que des opcodes Solana valides.

Cet exploit illustre pourquoi la sécurité blockchain doit aller au-delà des audits de surface et s'intéresser aux *patterns* architecturaux risqués. Sa sophistication le rend particulièrement redoutable dans l'écosystème Solana où la vitesse des transactions accélère les drainages.

B. Bundle Fragmenté + Bot de Trading (Market Manipulation)

Fonctionnement et Mécanismes d'Attaque

Cette attaque sophistiquée cible les pools de liquidités sur les DEX (tels que Raydium ou Orca) en exploitant la structure des marchés décentralisés et la psychologie des investisseurs. Elle combine des techniques avancées de fragmentation de transactions et d’intelligence artificielle pour créer une illusion de demande organique, permettant à l'attaquant de manipuler les prix à son avantage avant de liquider les fonds de manière abrupte.

1. Fragmentation des Transactions

L'attaquant utilise des **bundles** (groupes de transactions) pour diviser ses ordres en une multitude de micro-transactions. Cette approche permet :

- **D'éviter la détection** par les systèmes anti-front-running, qui surveillent généralement les grosses transactions suspectes.
- **De masquer l'origine des achats**, en répartissant les volumes sur plusieurs blocs et plusieurs wallets contrôlés par le bot.
- **D'optimiser l'impact sur le prix** en saturant le marché d'ordres achat soigneusement espacés, simulant une montée progressive de la demande.

La bibliothèque **solana-bundle-tools** est souvent utilisée pour orchestrer cette fragmentation, permettant d'envoyer des centaines de transactions en quelques secondes tout en minimisant les coûts de gas.

2. Bot de Trading Algorithmique

Pour rendre l'attaque crédible, le bot de trading est conçu pour imiter le comportement des investisseurs réels en utilisant des techniques de **machine learning** :

- **Modélisation des schémas d'achat** : Le bot analyse les données historiques de trading pour reproduire des motifs réalistes (timing aléatoire, volumes variables, ajustement dynamique du slippage).
- **Simulation d'activité humaine** : Il introduit des délais aléatoires entre les transactions et varie légèrement les montants pour éviter les motifs détectables par les algorithmes de surveillance.
- **Réaction aux conditions de marché** : Si le prix commence à stagner, le bot peut augmenter progressivement les volumes pour maintenir la pression acheteuse.

3. Phase de Pump and Dump

Une fois que le bot a artificiellement fait monter le prix (phase de **pump**), l'attaquant passe à la phase critique de liquidation (**dump**) :

- **Vente massive coordonnée** : Le bot exécute une série de sell orders groupées pour vider sa position en une fois, provoquant un effondrement du prix.
- **Optimisation des profits** : Les fonds sont souvent retirés via des bridges cross-chain ou convertis en stablecoins pour éviter le tracking on-chain.
- **Disparition des liquidités** : Dans certains cas, l'attaquant retire également les fonds du pool de liquidités, rendant le token impossible à vendre (rugpull classique).

4. Outils et Technologies Utilisés

- **Fragmentation des TX** :
  `solana-bundle-tools` pour grouper et envoyer les transactions.
  Stratégies de priorisation des frais pour garantir l'exécution.
- **Bot de Trading** :
  Python + `solana-py` pour interagir avec la blockchain.
  Modèles de ML simples (réseaux de neurones/reinforcement learning) pour générer des ordres réalistes.
  APIs comme Birdeye ou Jupiter pour l'analyse de marché en temps réel.
- **Dissimulation** :
  Utilisation de wallets jetables (burner wallets).
  Mixage de fonds via des protocoles privacy (ex : Sanctum).

5. Détection et Prévention

Pour contrer ce type d'attaque, les projets DeFi doivent :

- **Surveiller les motifs de trading** (ex : séries de micro-transactions depuis des wallets liés).
- **Implémenter des mécanismes de retard de liquidation** (timelocks) pour les gros volumes.
- **Analyser les bundles** pour identifier les transactions coordonnées.
- **Éduquer la communauté** sur les signaux d'alerte (hausse brutale du volume sans news fondamentales).

Cette méthode démontre comment les attaquants exploitent les failles structurelles des DEX, combinant ingénierie financière et automatisation pour des manipulations à grande échelle. Une défense efficace nécessite à la fois des outils techniques et une vigilance accrue des utilisateurs.

1.3 Importance de la Recherche Offensive

Ce projet vise à combler trois lacunes critiques de l'écosystème :

Audits incomplets : Les tests classiques négligent les backdoors "authority-based" et les attaques par bundles.
Détection naïve : Les outils surveillant les grosses transactions ignorent les micro-TX fragmentées.
Éducation technique : La majorité des développeurs Solana ne comprennent pas les risques des upgradeable programs.

Public visé :

- **Pentesters** : Méthodologie d'exploitation reproductible.
- **Développeurs DeFi** : Patterns anti-rugpull à intégrer (ex : timelock sur les changes d'authority).
  Auditeurs : Checklist pour détecter les instructions masquées.

1.4 Stack Technique Utilisée

Composant	Outil/Technologie
Exploit Development	Rust (Solana Program), Anchor Framework
Bundle Simulation	solana-remote-wallet + Tenderly
Bot Trading	Python (asyncio, solana-py)
Analyse On-Chain	Birdeye, Dune Analytics

2. Mécanismes et Psychologie de l'exploit via Update Authority (CVE-XXXX-XXXX) : 🔍

2.1 Anatomie d'un Rugpull

Un rugpull est une attaque **hybride**, combinant des **mécanismes techniques avancés** et une **manipulation psychologique** soigneusement orchestrée.

Phase 1 : Préparation (Ingénierie Sociale & Infrastructure Malveillante)

Création d'une façade légitime :
Site Web professionnel (souvent copié sur un projet existant).
Whitepaper technique, mais volontairement obscur pour dissimuler les backdoors.
Comptes réseaux sociaux automatisés (bots Twitter/Telegram).

Développement du smart contract :
Intégration discrète d'une fonction update_authority.
Utilisation de librairies vulnérables (ex : solana-program-library sans vérification des signers).

Phase 2 : Lancement (Initial Trust Building)

Déploiement du token :
Tokenomics trompeuses (ex : "taxes de redistribution" masquant un drain potentiel).
Pool de liquidité verrouillé (mais avec des droits d'administration cachés).

Marketing agressif :
Faux partenariats (mentions de projets connus sans accord).
Campagnes payantes via influenceurs complices.

Phase 3 : Croissance (Market Manipulation)

Manipulation du prix :
Bots de trading simulant un volume organique (via des **bundles fragmentés** pour éviter la détection).
"Pumps" artificiels déclenchés à intervalles réguliers pour alimenter le FOMO.

Communauté fictive :
Bots Discord/Telegram postant des "success stories".
Fausses annonces de listings imminents sur des CEX.

Phase 4 : Exécution (Exploit Technique Final)

Activation de la backdoor :
Appel de la fonction update_authority pour transférer la propriété du pool.
Drainage des fonds en quelques blocs (grâce à la faible latence de Solana).

Disparition :
Suppression des canaux de communication.
Blur des fonds via des mixers ou bridges cross-chain.

2.2 Psychologie de l'Arnaque (Social Engineering Breakdown)

Les attaquants exploitent des **biais cognitifs** critiques :

Biais d'ancrage : Les premières victimes voient des gains de +1000%, fixant une référence irréaliste.
Preuve sociale artificielle : Des centaines de "holders" bots (ex : wallets à 1 SOL) simulent l'adoption.
Aversion à la perte : Les victimes rechignent à vendre à perte malgré les red flags.
Effet de halo : Un audit factice (ex : logo "audité par X" sans rapport) suffit à rassurer.

Technique avancée :

> L'exploit CVE-XXXX-XXXX repose sur une **défaillance de vérification des permissions** dans les smart contracts Solana, permettant un **takeover silencieux**. Contrairement aux hacks "bruyants", cette méthode laisse peu de traces avant l'exfiltration.

2.3 Typologie des Rugpulls (Classification Offensive)

Type	Mécanisme Technique	Détection Évasion
Hard Rugpull	Backdoor via `update_authority`	Audit statique (analyse des CPI calls)
Soft Rugpull	Vente massive par les devs	Surveillance des wallets OTC
Liquidity Rugpull	Retrait du pool via `remove_liquidity`	Alertes on-chain (ex : Birdeye)
Honeypot	`transfer_restricted` dans le contrat	Tests de sell avant achat

Key Insight :

> Les rugpulls de type **hard** (comme notre exploit) sont les plus dangereux : ils nécessitent une **analyse dynamique** des transactions (ex : suivi des changements d'authority via Solscan).

---

3. Architecture du Projet 🏗️

1 Vue d'Ensemble Technique

Le projet utilise une architecture sophistiquée en plusieurs couches pour démontrer un rugpull réaliste. Cette complexité reflète ce que les attaquants utilisent réellement pour tromper leurs victimes. J'ai utilisé la **Blockchain** Solana car elle fournit une infrastructure décentralisée. Le choix de Solana n'est pas anodin - sa popularité et ses caractéristiques techniques en font une cible privilégiée. Parce qu'elle est rapide via transaction sur des blocs courts ne possède pas beaucoup de frais de transaction comparé a Ethernium ou Bitcoin (layer 1/2). C'est la blockchain la plus utilisé pour les investisseurs ou nouveaux arrivants dans l'univers des cryptomonnaies.
Les **Smart Contracts** utilisé par le framework Anchor pour créer des programmes sont apparemment légitimes. Anchor est choisi car il est le standard de l'industrie, ajoutant une couche de crédibilité. Afin d'organiser et d'articuler **l'application** : J'ai utilisé des scripts JavaScript/TypeScript pour interagir avec la blockchain. Ces scripts automatisent les opérations complexes nécessaires au rugpull. **Interface** : Script Python fournissant une interface interactive. Cette abstraction permet de démontrer le processus sans nécessiter une expertise technique approfondie.

2 Choix Technologiques et Justifications

Rust pour les Smart Contracts, c'est langage système offrant performance et sécurité. Ironiquement, ces qualités sont détournées pour créer des programmes malveillants efficaces. J'ai utilisé du typescript pour l'Interaction, cela m'a permis une manipulation précise de la blockchain avec un typage fort réduisant les erreurs. Python pour l'orchestration offre une interface simple pour démontrer des concepts complexes, rendant l'éducation accessible. Docker pour l'environnement assure la reproductibilité et l'isolation, essentielles pour une démonstration sécurisée.

3 Flux de Données et Interactions

Le système fonctionne selon un flux précis :

- L'utilisateur interagit avec le menu Python
- Python exécute les scripts JavaScript appropriés
- JavaScript communique avec les programmes Rust sur Solana

- Les programmes manipulent les états on-chain
- Les résultats remontent à travers les couches

Cette architecture multicouche est complexe, reflétant la sophistication des attaques réelles.

4 Technologies Utilisées

Composant	Technologie	Version	Rôle
Blockchain	Solana	1.18.26	Infrastructure décentralisée
Framework Smart Contract	Anchor	0.29.0	Développement de programmes Solana
Langage Smart Contract	Rust	1.75.0	Implémentation des programmes
Runtime Frontend	Node.js	20.19.2	Exécution des scripts d'interaction
Interface CLI	Python	3.x	Menu interactif et orchestration
Containerisation	Docker	Latest	Isolation et portabilité

2.3 Structure des Fichiers du projet

```
projet-rugpull/
├── programs/
│   └── demo_devnet/
│       ├── src/
│       │   ├── lib.rs              # Programme principal
│       │   ├── lib_save_normal.rs  # Version légitime sauvegardée
│       │   └── lib_save_attack.rs  # Version malveillante
│       └── Cargo.toml
├── tests/
│   ├── cloneToken.js              # Clonage de token
│   ├── setupVault.js              # Configuration du vault
│   ├── addLiquidity.js            # Ajout de liquidité
│   ├── makeTokenCredible.js       # Amélioration crédibilité
│   ├── bundle.js                  #Créations des bundles fragmenté
│   ├── exploitDrain.js            # Exécution du rugpull /soit via upgrade
authority ou via le swap des bundles en solana. (en vérité ces actions sont dans
des scripts séparer, mais plus de simplicité je ne les ai pas listés)
│   └── ...
├── solana_trading_bot/
│   └── bot.js                     # Bot de trading simulé
├── rugpull_realistic.py           # Interface principale
├── Dockerfile                     # Configuration Docker
└── requirements.txt               # Dépendances Python
```

---

3. Analyse du Code Rust : Version Normale vs Malveillante 🔍

3.1 Principe de l'Update Authority

Sur Solana, les programmes déployés peuvent avoir une **update authority** qui permet de :

- Modifier l code du programme après déploiement
- Ajouter ou retirer des fonctionnalités
- Introduire des backdoors

Cette fonctionnalité, conçue pour les mises à jour légitimes, devient une vulnérabilité majeure si elle n'est pas révoquée.

Analyse Technique Détaillée de l'Exploit Solana

1. Programme Normal (lib_save_normal.rs)

```
pub fn withdraw(ctx: Context<Withdraw>, amount: u64) -> Result<()> {
    // Logique légitime de retrait
    let vault = &ctx.accounts.vault;
    let seeds = &[
        b"vault_authority",
        vault.token_mint.as_ref(),
        &[vault.bump]
    ];
    let signer = &[&seeds[..]];

    let transfer_instruction = Transfer {
        from: ctx.accounts.vault_token_account.to_account_info(),
        to: ctx.accounts.user_token_account.to_account_info(), // Destination
utilisateur légitime
        authority: ctx.accounts.vault_authority.to_account_info(),
    };

    let cpi_ctx = CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        transfer_instruction,
        signer,
    );

    token::transfer(cpi_ctx, amount)?;
    Ok(())
}
```

Points Sécurisés :

Les **points sécurisés** de ce code montrent trois bonnes pratiques essentielles sur Solana :

- **Signature par PDA** : Le programme utilise vault_authority, une clé dérivée cryptographiquement (via des seeds comme le mint du token et un bump), pour signer les transactions. Cela évite de stocker une clé privée sensible et suit le standard Solana.
- **CPI sécurisé** : Le transfert est exécuté via le programme SPL Token (CPI), un contrat audité et non modifiable, ce qui garantit que la logique de transfert est fiable.
- **Contrôle des entrées** : Bien que le montant (amount) soit choisi par l'utilisateur, il est automatiquement limité par le solde disponible dans le vault, empêchant un drain excessif.

Ces mécanismes assurent que la fonction withdraw est **sûre et décentralisée**, à condition qu'aucune backdoor ne soit cachée ailleurs dans le contrat...

---

2. Programme Malveillant (lib_save_attack.rs)

```
pub fn drain_vault(ctx: Context<DrainVault>) -> Result<()> {
    let vault = &ctx.accounts.vault;
    let amount = ctx.accounts.vault_token_account.amount; // Prend TOUS les fonds

    let seeds = &[
        b"vault_authority",
        vault.token_mint.as_ref(),
        &[vault.bump]
    ];
    let signer = &[&seeds[..]];

    let transfer_instruction = Transfer {
        from: ctx.accounts.vault_token_account.to_account_info(),
        to: ctx.accounts.attacker_token_account.to_account_info(), // Adresse de
l'attaquant en dur !
        authority: ctx.accounts.vault_authority.to_account_info(),
    };

    let cpi_ctx = CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        transfer_instruction,
        signer,
    );

    token::transfer(cpi_ctx, amount)?;
    Ok(())
}
```

Mécanisme de l'Exploit :

Fonction Fantôme :

Le programme semble normal avec une fonction withdraw classique, mais cache une fonction fantôme drain_vault . Contrairement à withdraw où l'utilisateur contrôle le montant, drain_vault vide tout le solde du vault sans demander de permission. L'astuce repose sur trois manipulations :

La première repose sur un backdoor cachée. L'adresse de destination (attacker_token_account) est codée en dur dans le programme, comme une clé secrète que seul l'attaquant connaît.

Il y a ensuite une "usurpation d'identité" dans le code. La fonction utilise la même autorité (vault_authority) que les opérations légitimes, comme si un voleur utilisait les clés du propriétaire pour vider la maison. Autrement dit nous avons le contrôle total des tokens qui seront créer a partir de ce smart contrat. Et ceci peut importe la durée de vie des tokens créés.

Le transfert passe par le programme SPL Token (standard), donc les outils voient une transaction "normale". Comme la fonction n'est pas documentée (absente de l'IDL), même un audit pourrait la manquer.

Exemple Simplifié :
Imaginez un contrat qui dit : *"Si Jean appelle `withdraw`, il retire 10€. Mais si quelqu'un appelle `drain_vault` (même Jean), tout l'argent part vers mon compte perso, sans trace."*

Pourquoi c'est dangereux ?

Les utilisateurs voient seulement withdraw et pensent que le contrat est sûr.
L'attaquant peut déclencher drain_vault à tout moment, sans avertissement.

Le problème qui en fait un technique redoutable :

Le "update authority" est la clé qui permet de toujours avoir le contrôle en restant furtif. En effet nous pouvons facilement créer des tokens solana a partir d'un smart contrat clean (sans backdoor). Nos tokens seront réellement clean et puisqu'ils n'auront aucune contrefaçon et le resterons dans le temps. Jusqu'a ce que nous utilisons "update authority". N'ayant pas révoquée l'authority sur les tokens créé, ils restent lié en temps réel au smart contrat déployer sur la blockchain. Nous avons donc encore la possibilité en temps que développeur du smart contrat de faire un "update authority" autrement dit, une mise a jour du smart contrat déployer sur la blockchain solana. C'est via cette mise a jour que nous allons modifié le code du smart contrat dont déjà plusieurs tokens (memecoins/fake projet) dépendent. En ajoutant une fonction pour drainer la liquidité des tokens. Cette technique est juste monstreuse ! Furtif et très rapide !

📜 Article du Nasdaq USA décrivant une attaque en 2021 :
Squid Game (SQUID), la crypto-monnaie inspirée du succès de Netflix, s'est effondrée hier, laissant de nombreux investisseurs sans rien. La mème pièce a été lancée le 26 octobre et a gagné plus de 23 000 000 % en une semaine, selon les données de CoinMarketCap. Elle a culminé à environ 2 862 dollars avant de chuter à une fraction de centime en quelques minutes. [...] Dans ce cas, les escrocs anonymes ont emporté environ 3,4 millions de dollars.

Récapitulatif des attaques blockchains et le montant volée par les Hackers :

Projet	Date	Montant Volé	Méthode	Leçons
AnubisDAO	2021	$60M	Liquidity removal	Vérifier les multisigs
Squid Game Token	2021	$3.4M	Anti-sell mechanism	Tester les ventes
Luna Yield	2022	$6.7M	Mint exploit	Auditer le code
Fintoch	2023	$31.6M	Exit scam	KYC team

Comment s'en protéger ? ⚡

Toujours vérifier toutes les fonctions d'un programme (même celles non documentées).
Utiliser des outils comme Solscan pour tracer les appels suspects.

Mais dans le cas d'un update authority aucune méthode ne permet de sans protéger, si ce n'est de ne pas investir votre argent dans un token solana ayant "autority" non révoqué. Le soucis est que certain token solana ayant une "authority" non révoqué sont des tokens s'appuyant sur des vrais projets crypto. Problème vous ne pourrait pas faire la différence vous, en tant que simple investisseur, entre le vrai du faux ! 🚨

---

4. Vecteurs d'Attaque JavaScript 💻

4.1 Script Principal Python (rugpull_realistic.py)

Le script Python orchestre l'ensemble du processus avec une interface interactive :

PYTHON

```python
def show_menu(custom_menu_items=None):
    """Affiche le menu principal interactif avec navigation par flèches"""
    menu_items = [
        {"id": "1", "text": "Vérifier les prérequis", "function":
"check_prerequisites"},
        {"id": "2", "text": "Clonage de token", "function": "clone_token"},
        {"id": "3", "text": "Ajout de liquidité", "function": "add_liquidity"},
        {"id": "4", "text": "Amélioration de la crédibilité", "function":
"make_token_credible"},
        {"id": "5", "text": "Bot de trading", "function": "execute_bot_trading"},
        # ...
    ]
```

Ce script Python implémente une interface en ligne de commande interactive pour orchestrer un processus complexe, lié à la création et la manipulation de tokens cryptographiques. La fonction show_menu() sert de hub central avec un système de navigation intuitif utilisant les flèches du clavier. Le menu propose cinq options principales :

1. Vérification des prérequis techniques,
2. Clonage de token (création d'une copie),
3. Ajout de liquidité (pour rendre le token échangeable),
4. Amélioration de la crédibilité (simulation d'intérêt légitime)
5. Bot de trading (automatisation des transactions).

Chaque option est associée à une fonction spécifique comme `clone_token()` ou `execute_bot_trading()`. L'architecture modulaire permet d'ajouter facilement de nouvelles fonctionnalités via le paramètre `custom_menu_items`. La structure des éléments du menu (dictionnaires avec id, texte et fonction) montre une conception orientée objet pour une maintenance simplifiée. Ce type d'interface est particulièrement utile pour les opérations blockchain complexes nécessitant plusieurs étapes séquentielles. Le système permet à un utilisateur non technique d'exécuter des processus sophistiqués sans avoir à écrire du code manuellement.

4.2 Scripts JavaScript d'Interaction

cloneToken.js - Création du token :

```JS
async function cloneExistingToken() {
    const connection = new Connection(clusterApiUrl(SOLANA_NETWORK), 'confirmed');

    // Récupération des métadonnées du token source
    const sourceMint = new PublicKey(SOURCE_TOKEN_ADDRESS);
    const sourceMintInfo = await getMint(connection, sourceMint);

    // Création du nouveau token avec les mêmes paramètres
    const newMint = await createMint(
        connection,
        wallet,
        wallet.publicKey,
        wallet.publicKey,
        sourceMintInfo.decimals
    );

    // Mint initial de tokens
    await mintTo(
        connection,
        wallet,
        newMint,
        tokenAccount.address,
        wallet.publicKey,
        10000000 * Math.pow(10, sourceMintInfo.decimals)
    );
}
```

Ce script JavaScript/Node.js permet de cloner un token existant sur la blockchain Solana en reproduisant ses caractéristiques techniques. La fonction principale cloneExistingToken() commence par établir une connexion au réseau Solana via l'API publique. Elle récupère d'abord les métadonnées du token source (comme son adresse et le nombre de décimales) à partir de son adresse publique (SOURCE_TOKEN_ADDRESS). Ensuite, le script crée un nouveau token en utilisant la fonction createMint(), en copiant spécifiquement le nombre de décimales du token original pour garantir la compatibilité.

Le wallet connecté devient à la fois l'autorité de mint (création) et l'autorité de freeze (gel). Après cette création, le script effectue un mint initial de 10 millions de tokens (en tenant compte des décimales) vers un compte token associé. Ce processus utilise la fonction mintTo() qui nécessite la signature du wallet propriétaire. L'architecture modulaire du code permet de facilement adapter le token cloné en modifiant des paramètres comme le nombre de décimales ou la quantité mintée.

makeTokenCredible.js - Amélioration de la crédibilité :

```JS
// Brûlage de tokens pour réduire la supply
if (BURN_PERCENTAGE > 0) {
    const burnAmount = Math.floor(currentBalance * BURN_PERCENTAGE / 100);
    await burn(
        connection,
        wallet,
        tokenAccount.address,
        tokenMint,
        wallet.publicKey,
        burnAmount
    );
}

// Révocation des autorités pour paraître légitime
if (REVOKE_MINT_AUTHORITY) {
    await setAuthority(
        connection,
        wallet,
        tokenMint,
        wallet.publicKey,
        AuthorityType.MintTokens,
        null // Révocation
    );
}
```

Ce script **makeTokenCredible.js** sert à rendre un token plus "crédible" aux yeux des utilisateurs ou investisseurs, en appliquant deux techniques souvent utilisées sur Solana.

1. **Brûler des tokens ("burn")** :
   Le code vérifie si un pourcentage de tokens doit être brûlé (BURN_PERCENTAGE > 0). Si c'est le cas, il calcule le montant à brûler en prenant ce pourcentage du solde actuel, puis il appelle la fonction burn() pour détruire ces tokens, c'est-à-dire les retirer de la circulation. Cela permet de réduire l'offre totale, ce qui peut donner l'impression que le token a plus de valeur ou est géré de manière sérieuse.

2. **Révocation de l'autorité de mint** :
   Ensuite, s'il est demandé (REVOKE_MINT_AUTHORITY activé), le script utilise setAuthority() pour supprimer l'autorité de mint (MintTokens). Cela veut dire que personne, même le créateur, ne pourra créer de nouveaux tokens à l'avenir, ce qui donne plus de confiance aux utilisateurs, car cela empêche la création de tokens supplémentaires de manière cachée (ce qui serait perçu comme un rug pull potentiel).

Mais on va laisser "authority update" accessible.

Étape : Phase d'Accumulation (bot.js) 🐍

Durant cette période, l'attaquant :

- Utilise des bots pour simuler une activité de trading
- Crée du contenu marketing (faux partenariats, roadmap ambitieuse)
- Encourage les dépôts dans le vault avec des promesses de rendements

Indicateurs : Volume de trading artificiel, croissance "organique" du nombre de holders, buzz sur les réseaux sociaux.

Si l'on utilise la méthode de l'update authority :

Étape : Mise à Jour Malveillante 🧪

(bot.js - détailler après)
C'est le moment critique où le piège se referme. L'attaquant utilise son autorité de mise à jour pour :

- Remplacer le programme légitime par une version malveillante
- Ajouter une fonction permettant de drainer tous les fonds
- Maintenir les fonctions normales pour ne pas éveiller les soupçons

Technique : La mise à jour est instantanée sur Solana. Il n'y a pas de période de grâce ou d'alerte pour les utilisateurs.

Étape : Exécution du Drainage 🔥

L'exécution finale est rapide et dévastatrice :

- Appel de la fonction de drainage nouvellement ajoutée
- Transfert de tous les tokens vers le wallet de l'attaquant
- Retrait de la liquidité des DEX
- Effacement de toute trace (sites web, réseaux sociaux)

Durée : L'ensemble du processus de drainage prend généralement moins d'une minute.

Si on utilise la méthode de manipulation de marché via bundles :

Etape : Manipulation de Marché et Bundles 🧩

Si l’on utilise une **stratégie de manipulation de marché via bundles**, la phase de mise à jour malveillante devient inutile. L’attaquant contourne la modification du smart contract et agit directement sur les échanges en chaîne. Voici les étapes clés :

Cette méthode sophistiquée de manipulation de marché sur Solana repose sur une préparation minutieuse et l'utilisation stratégique des fonctionnalités blockchain. L'attaquant commence par créer une armée de wallets jetables (100+), alimentés discrètement en SOL via des mixers pour brouiller les pistes. La phase clé consiste à simuler une activité organique en exécutant des swaps circulaires entre ces wallets, créant artificiellement du volume et de la liquidité apparente.

Grâce au système de bundles de Solana, ces milliers de micro-transactions sont regroupées en blocs coordonnés, optimisant les frais tout en évitant les alertes des systèmes de surveillance. Des bots sophistiqués synchronisent ces opérations pour amplifier l'effet FOMO, déclenchant des vagues d'achats à des intervalles précis qui semblent naturels. Lorsque suffisamment de victimes ont investi, l'attaquant exécute en quelques secondes un swap massif des tokens frauduleux contre les SOL du pool, en utilisant là encore des bundles pour masquer l'opération.

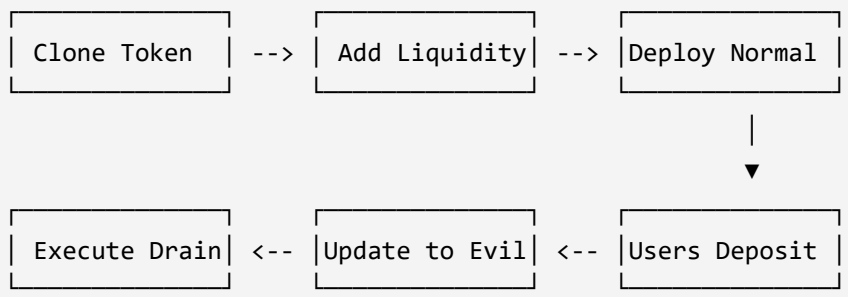
L'avantage décisif de cette technique réside dans son absence d'interaction avec les smart contracts - tout se passe au niveau des échanges décentralisés, rendant l'attaque indétectable par les outils d'audit traditionnels. Les fonds sont ensuite blanchis via des bridges cross-chain comme Wormhole, tandis que les wallets utilisés sont abandonnés, laissant peu de traces exploitables pour l'analyse forensique.

Configuration Docker

Le Dockerfile assure un environnement reproductible :

```
FROM ubuntu:22.04

# Installation des dépendances système
RUN apt-get update && apt-get install -y \
    python3 python3-pip curl git build-essential

# Installation Node.js
RUN curl -fsSL https://nodejs.org/dist/v20.19.2/node-v20.19.2-linux-x64.tar.xz | \
    tar -xJ -C /usr/local --strip-components=1

# Installation Solana CLI
RUN sh -c "$(curl -sSfL https://release.anza.xyz/v1.18.26/install)"

# Installation Rust et Anchor
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
RUN cargo install --git https://github.com/coral-xyz/anchor anchor-cli --locked
```

Diagramme de Flux



🤖 Bot de Trading Solana (`bot.js`)

🌐 Aperçu

```
import { Connection, PublicKey, Keypair, Transaction, sendAndConfirmTransaction,
ComputeBudgetProgram, VersionedTransaction } from '@solana/web3.js';
import { getOrCreateAssociatedTokenAccount, getAccount, TOKEN_PROGRAM_ID,
ASSOCIATED_TOKEN_PROGRAM_ID } from '@solana/spl-token';
import fs from 'fs';
...

dotenv.config();

# Configuration
const RPC_ENDPOINT = process.env.RPC_ENDPOINT || 'https://api.mainnet-
beta.solana.com';
const TOKEN_ADDRESS = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
const WSOL_ADDRESS = 'So111111111111111xxxxxxxxxxxxxxxxxxxxxxxx'; // Wrapped SOL
const WALLETS_FILE = './wallets.json';
const SLIPPAGE_BPS = 500; // 1% = 100 basis points
const JUPITER_API = 'https://quote-api.jup.ag/v6';

class SolanaTradingBot {
    constructor() {
        this.connection = new Connection(RPC_ENDPOINT, 'confirmed');
        this.tokenMint = new PublicKey(TOKEN_ADDRESS);
        this.wsolMint = new PublicKey(WSOL_ADDRESS);
        this.wallets = [];
        this.solPrice = 0;
        this.tokenDecimals = 6; // Valeur par défaut, sera mise à jour

# Statistiques
        this.startTime = Date.now();
        ...
```

Ce fichier implémente un bot de trading sophistiqué (à but éducatif) pour Solana, simulant des stratégies de manipulation de marché avec plusieurs portefeuilles. Conçu pour la démonstration et l'apprentissage, il interagit avec la blockchain Solana, gère des portefeuilles, exécute des trades (achat/vente) et simule des comportements avancés pour analyser leur impact.

Fonctionnalités & Composants Clés

1. Configuration & Initialisation

Variables d'environnement : Charge l'endpoint RPC et autres paramètres depuis `.env`.
Configuration des tokens : Définit le token cible et l'adresse du Wrapped SOL (WSOL).
Gestion des portefeuilles : Charge ou génère 20 portefeuilles (keypairs), stockés dans

`wallets.json`.
Données de marché : Récupère le prix du SOL et les infos du token (décimales, symbole) via des APIs externes.

2. Classe Principale : `SolanaTradingBot`

Encapsule toute la logique du bot.

📊 Variables d'état portefeuilles :

Portefeuille : Tableau d'objets suivants les balances SOL et tokens.
Métriques de marché : Nombre de trades, ratio achat/vente, volume (USD), market cap simulé.
Stratégies : Suivi de l'impact des manipulations de marché.
Motifs spéciaux : Indicateurs de conditions anormales (FOMO, dip, etc.).
Multiplicateur de volume : Augmente artificiellement le volume affiché (effet démo).

3. Fonctionnalités Principales

Fonctionnalités Principales (Analyse Technique)

Ce système Python intègre une suite complète d'outils pour interagir avec la blockchain Solana, combinant gestion de portefeuilles, analyse de marché et stratégies de trading avancées. Le module **A** gère les portefeuilles (chargement/génération aléatoire de clés et vérification des balances SOL/tokens). Le module **B** agrège des données on-chain (prix du SOL via CoinGecko) et off-chain (métadonnées de tokens via Jupiter API). La **logique de trading (C)** exécute des swaps via Jupiter, avec des algorithmes ajustant dynamiquement les montants selon les tendances détectées (FOMO, mouvements de baleines).

Les **stratégies avancées (D)** simulent des manipulations de marché complexes :

- *Trading circulaire* (artificielle inflation du volume)
- *Ordres superposés* (spoofing de l'order book)
- *Lavage de volume* (transactions bidon)
- *Effets psychologiques* (FOMO via achats coordonnés)

Le système inclut aussi des **motifs prédéfinis (E)** pour reproduire des scénarios de marché (ruées FOMO, corrections, accumulations "baleines"). Enfin, un **dashboard temps réel (F)** visualise l'activité via des statistiques dynamiques, un suivi des stratégies actives et un reporting post-trade. L'architecture modulaire permet d'ajouter facilement de nouvelles stratégies tout en maintenant une interface utilisateur intuitive.

Note : Ces fonctionnalités, bien que techniquement impressionnantes, peuvent servir à des fins éthiques (backtesting) ou malveillantes (market manipulation) selon leur usage.

Détails Techniques

- **Blockchain** : Utilise `@solana/web3.js` et `@solana/spl-token`.
- **APIs** : Jupiter (swaps) et CoinGecko (prix).
- **Terminal** : Affichage formaté avec `cli-table3` et `chalk`.
- **Gestion des Erreurs** : Robustesse face aux échecs d'API/transactions.
- Commentaires et explications intégrés.

Avertissements & Usage ⚠️

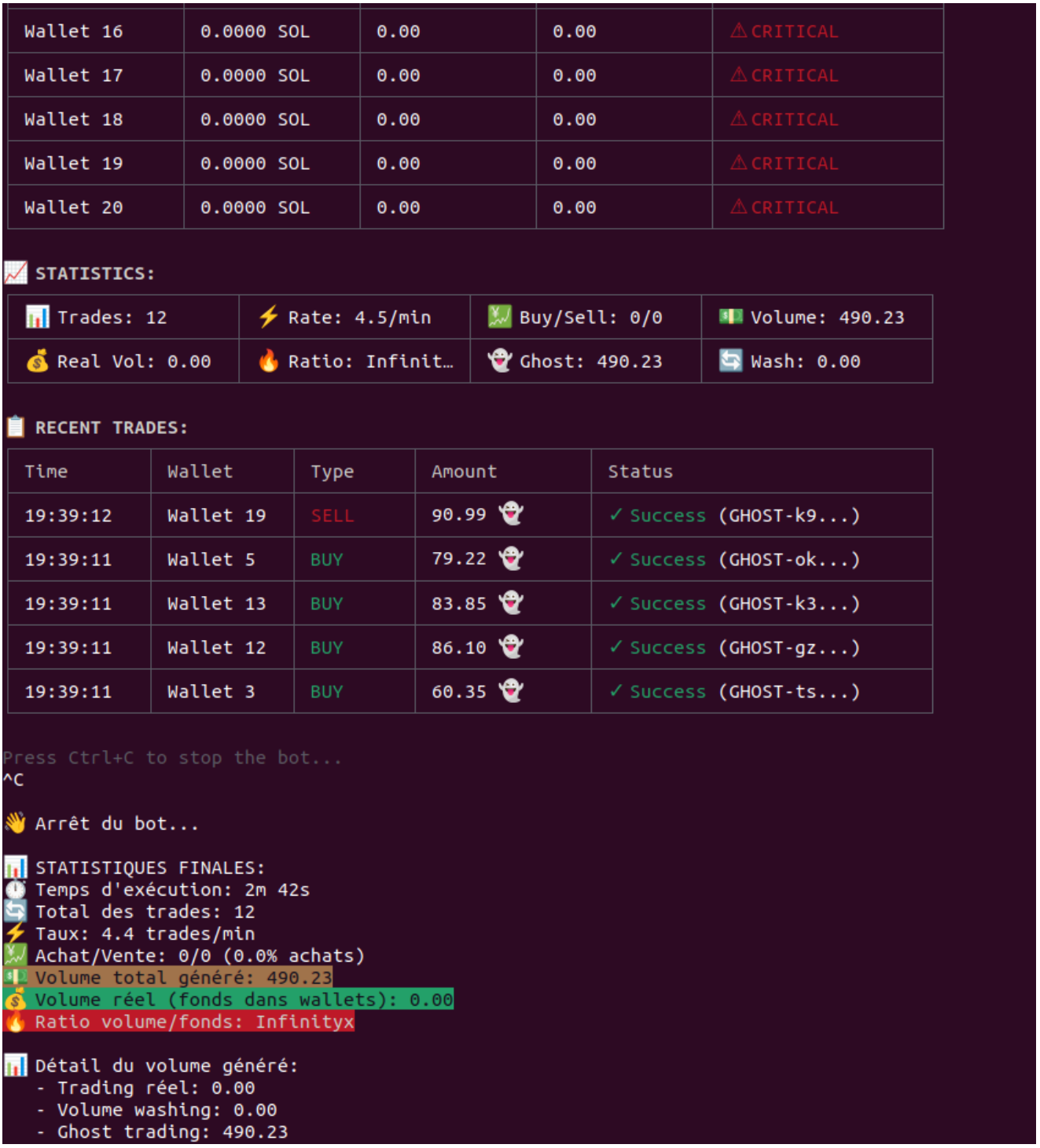
Éducatif Uniquement : Ne pas utiliser en réel.
Volume Artificiel : Multiplié pour la démo.
Fonds Requis : SOL de test nécessaire pour les simulations.

Ce bot est un outil complet pour étudier les mécanismes de marché et leurs manipulations. Idéal pour la recherche ou la formation en trading crypto.

```
# 🛠️ **Exécution du Bot**
1. 🔄 `npm install`
2. 📝 Configurez `.env`
3. 🚀 `node bot.js`
```

```
[+] Bot actif : Simulation FOMO en cours...
==> 📈 Volume artificiel : 1.2M$ (x10)
```

Capture d'écran :



Wallet 16	0.0000 SOL	0.00	0.00	⚠ CRITICAL
Wallet 17	0.0000 SOL	0.00	0.00	⚠ CRITICAL
Wallet 18	0.0000 SOL	0.00	0.00	⚠ CRITICAL
Wallet 19	0.0000 SOL	0.00	0.00	⚠ CRITICAL
Wallet 20	0.0000 SOL	0.00	0.00	⚠ CRITICAL

📈 STATISTICS:

📊 Trades: 12	⚡ Rate: 4.5/min	💹 Buy/Sell: 0/0	💵 Volume: 490.23
💰 Real Vol: 0.00	🔥 Ratio: Infinit…	👻 Ghost: 490.23	🔄 Wash: 0.00

📋 RECENT TRADES:

Time	Wallet	Type	Amount	Status
19:39:12	Wallet 19	SELL	90.99 👻	✓ Success (GHOST-k9...)
19:39:11	Wallet 5	BUY	79.22 👻	✓ Success (GHOST-ok...)
19:39:11	Wallet 13	BUY	83.85 👻	✓ Success (GHOST-k3...)
19:39:11	Wallet 12	BUY	86.10 👻	✓ Success (GHOST-gz...)
19:39:11	Wallet 3	BUY	60.35 👻	✓ Success (GHOST-ts...)

```
Press Ctrl+C to stop the bot...
^C

👋 Arrêt du bot...

📊 STATISTIQUES FINALES:
⏱️ Temps d'exécution: 2m 42s
🔄 Total des trades: 12
⚡ Taux: 4.4 trades/min
💹 Achat/Vente: 0/0 (0.0% achats)
💵 Volume total généré: 490.23
💰 Volume réel (fonds dans wallets): 0.00
🔥 Ratio volume/fonds: Infinityx

📊 Détail du volume généré:
   - Trading réel: 0.00
   - Volume washing: 0.00
   - Ghost trading: 490.23
```

Impact des vulnérabilités sur Solana :

Vulnérabilité	Description	Impact
Update Authority Active	Permet la modification du programme	Critique
Absence de Timelock	Mises à jour instantanées	Élevé
Pas de Multisig	Contrôle unilatéral	Élevé
Code Non Vérifié	Impossibilité d'audit	Moyen

Note :

La complexité technique de Solana crée une opacité qui favorise les attaquants :

- *Peu d'utilisateurs comprennent vraiment le fonctionnement*
- *Les outils d'analyse sont limités et techniques*
- *La vérification manuelle est pratiquement impossible*
- *Les audits sont coûteux et souvent superficiels*

💰 5. Démonstration du PoC (Proof of concept)

(TEST sur le DEVNET)

Réseau de test public utilisé par les développeurs pour tester leurs applications décentralisées ⚠️ *

Interface d'accueil du programme :

Dès que le programme démarre, nous arrivons sur une interface d'accueil qui décrit brièvement le but de l'outil.
Message : *Ce script simule une attaque complète exploitant l'autorité de mise à jour.*
Avec un `Disclaimer/Avertissement` que les gens comprennent que cette outil, en étant un peu modifier peut permettre de vole des vrais fonds sur la blockchain Solana.

Image 1 :

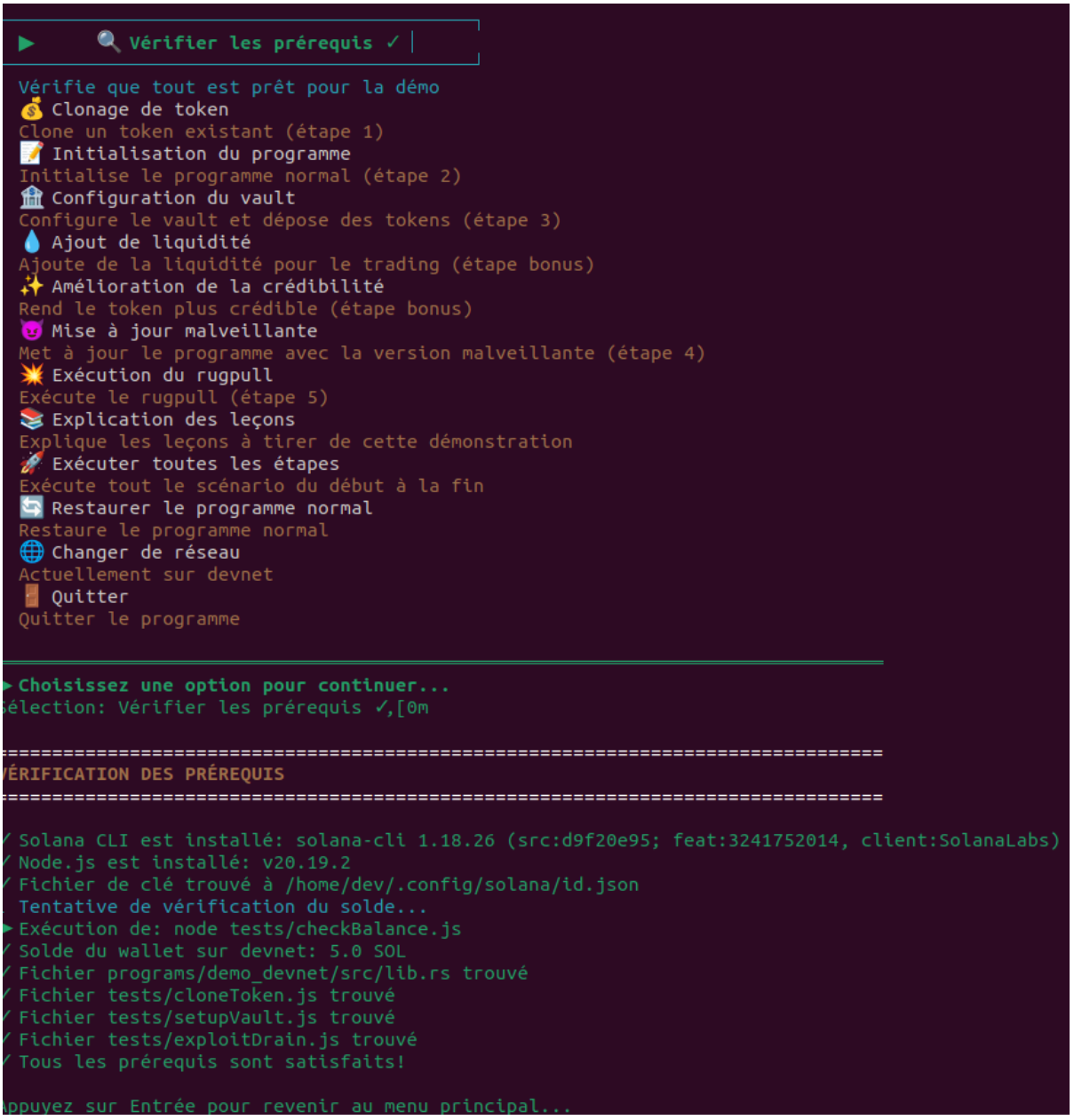
Interface du programme.

L'interface est un menu interactif qui comprend plusieurs onglets. Cette interface est géré par Python via la librairie : `colorama` .
La sélection des différentes étapes du programme ce fait par l'utilisation des flèches de direction du claviers.

Durant la première étape, le programme vérifie si les prérequis sont validé afin de poursuivre vers les étapes suivantes. Il regarde si : `Solana` et `Node` sont bien installé et que les versions correspondent a celle attendue. On analyse ensuite le système pour être sur qu'il contient bien un Wallet Solona : `/home/dev/.config/solana/id.json`.

Il vérifie que le wallet bien alimenté en Solana via un code JS : `CheckBalance.js`. La dernière vérification concerne les programmes JS qui serviront durant la suite du projet. Petite précision, est tant donné que nous somme sur le `Devnet` (Réseau Solana dédié aux développeurs) les Solana utilisé sont fictifs.

Image 2 :



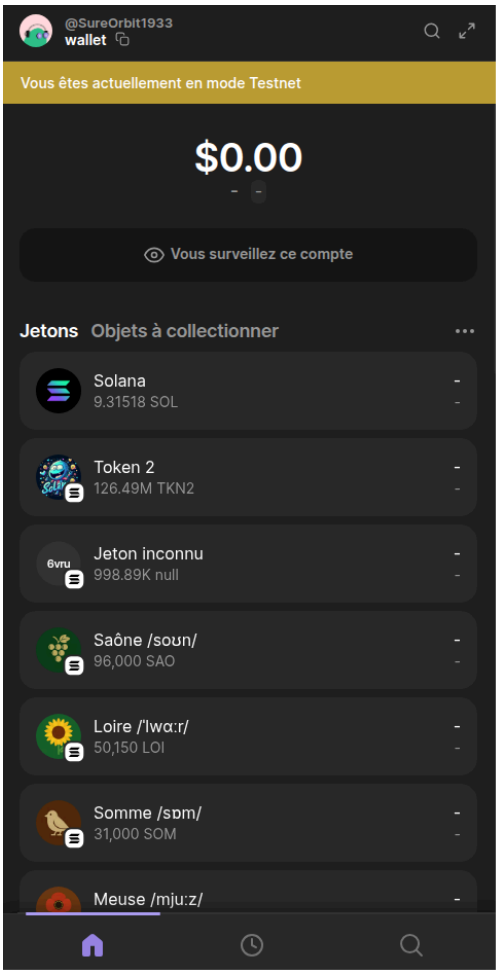
Clonage de Token :

L'étape du clonage de token est l'une des phase du projet qui m'a pris le plus de temps de développer. L'idée est de pourvoir réussir a copier tout les informations que possède un Token seulement grâce a son MINT (adresse).

Image 3 :

J'ai récupérer l'adresse du compte de quelqu'un sur la blockchain Solana (Devnet) afin qu'avoir un visuelle sur les tokens de cette personne. Puis je vais pouvoir en récupérer un pour le cloner.

Le Token choisi ⇒ `Token2`



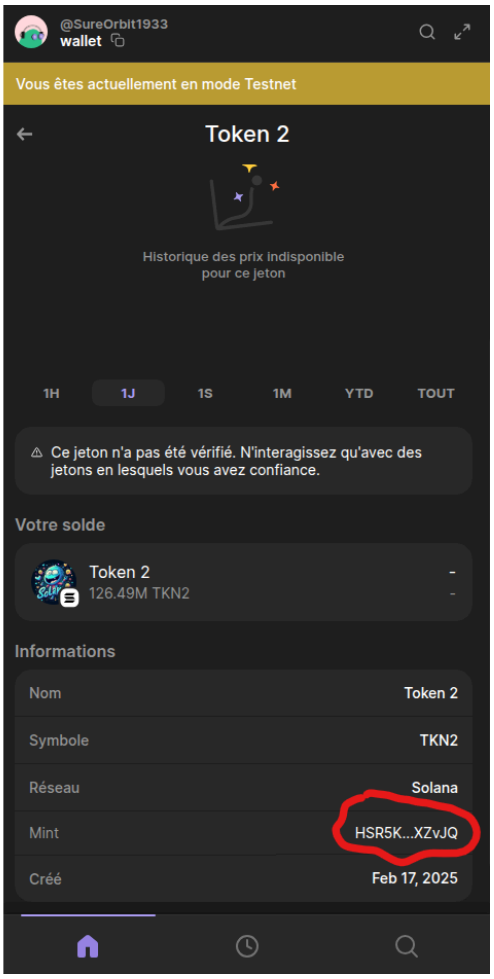


Image 4 :

Adresse du token sélectionner ⇒ `"HSR5K...XZvJQ"`

Passons a l'action !

Lorsque je lance l'étape de clonage de Token le programme me demande de rentrer l'adresse du Token a cloner (Son MINT). Nous avons juste lui donné et le programme python (code mère) va automatiquement transmettre l'adresse du Token au code JS qui va lancé le processus.

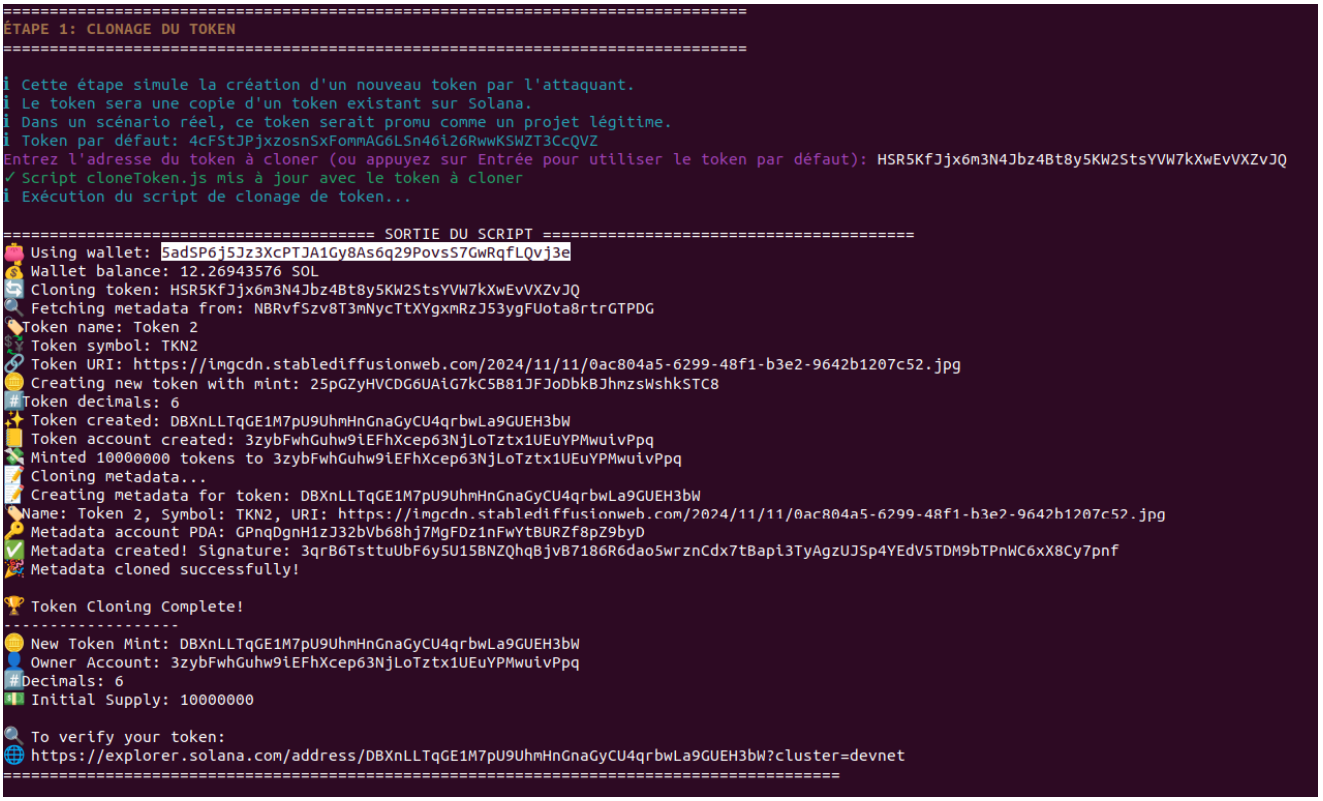


Image 5 :
Ce que va faire le code JS est :

Connexion à Solana: Se connecte au réseau (devnet) via l'API Solana.
Chargement du Wallet : Charge la clé privée (keypair: `000xxxxxxxxxxxxxxxx0`) de l'utilisateur pour signer les transactions.
Vérification du solde : Vérifie que le wallet a assez de SOL pour payer les frais de transaction.
Lecture du Token source : Récupère les informations du token à cloner :
(adresse mint: `"HSR5K...XZvJQ"`, métadonnées : `logo`, décimales : `nombre de jeton`, etc.).

Création du nouveau Token :

- Crée un nouveau mint avec les mêmes décimales que le token source : `6 decimal`
- Crée un compte associé pour recevoir les nouveaux tokens : `3zybFwXXXXXXXXXXXXXXXXXX`
- Mint le nombre désiré de tokens sur ce compte : `10000000 tokens`

Clonage des métadonnées : Si le token source a des métadonnées (nom, symbole, URI), elles sont copiées sur le nouveau token via le programme Metaplex.

Affichage des informations : Affiche l'adresse du nouveau mint, le compte propriétaire, le nombre de décimales, etc.

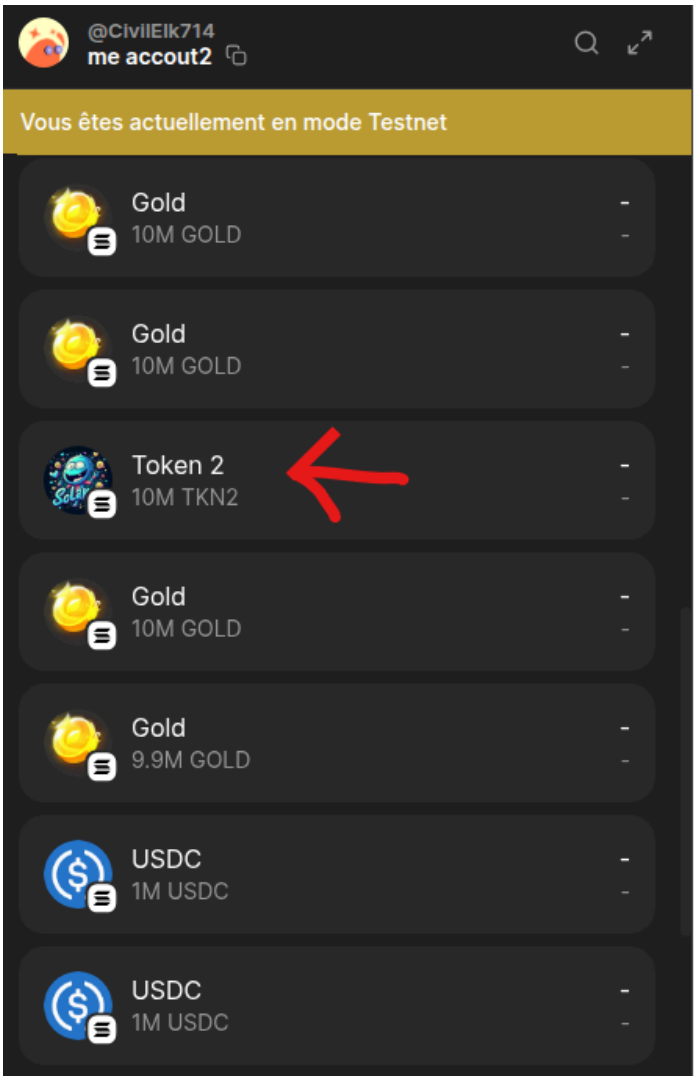


Image 6 :

Nous pouvons voir que le Token a bien été créé sur notre Wallet !

Image 7 :

Ce script simule la création d'un **vault** (coffre-fort) pour stocker le nouveau token, en utilisant un PDA (Program Derived Address) comme autorité du vault.

Ce que va faire le code JS :

Connexion et chargement du wallet : Comme précédemment, connexion à Solana et chargement de la clé privée.
Chargement de l'IDL Anchor : Charge la description du programme (IDL) pour interagir avec le smart contract déployé : `BoPqDQ5XXXXXXXXXXXXXX`
Calcul du PDA du vault : Calcule l'adresse spéciale (PDA) qui servira d'autorité pour le vault, dérivée du mint du token et de l'ID du programme.
Création des comptes de tokens :

- Crée un compte de token pour l'utilisateur (si besoin).
- Crée un compte de token pour le PDA du vault (vault token account).

Transfert de tokens : Transfère la quasi-totalité des tokens de l'utilisateur vers le vault (simulateur de dépôt): `3GhXhEVjhkShp9XXXXXXXX`
Mise à jour du script d'exploit : Met à jour l'adresse du vault dans le script exploitDrain.js pour préparer la démonstration de l'attaque.
Affichage des informations : Affiche les adresses et soldes des comptes impliqués.

🔄 Méthode de Mise à Jour de l'Autorité

Image 8 :

🎨 **Visuel/Analogie**

🏛️ *Coffre fort qui contient votre argent* - Smart contrat

🔧 *Les plans de constructions du coffre fort* - Code Rust

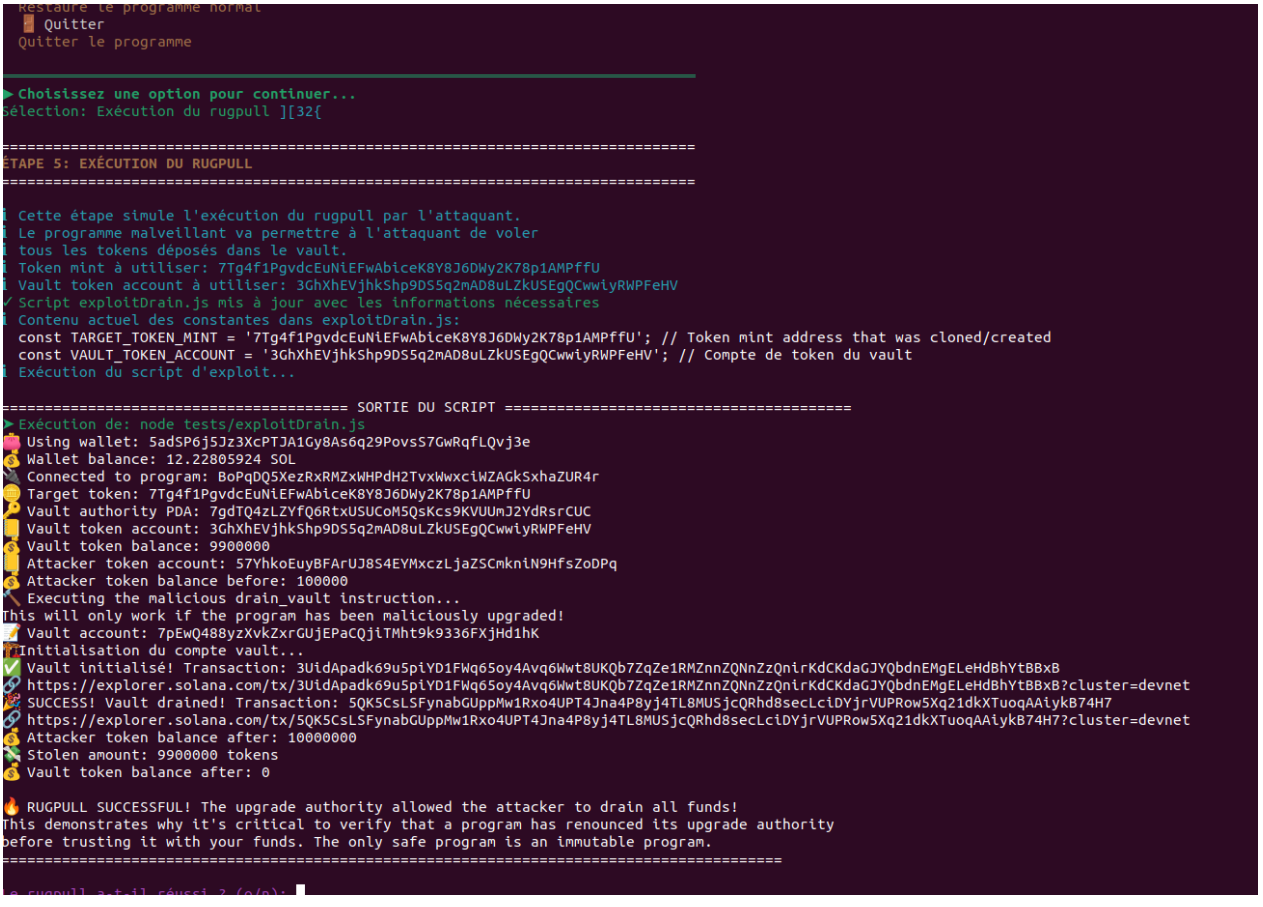
🔑 *La clé du coffre fort* - PDA authority

👴💻 *Le voleur va vous dire de mettre de l'argent la coffre fort en vous disant qu'il ne possède pas la clef pour l'ouvrir.* - Attractives tokens launch (with fake informations)

🚨 *Le voleur va revenir ouvrir le coffre fort est volé tout l'argent que les gens ont déposé* - Update authority & Steal money

Exécution de l'exploit

- Ce script simule une attaque où un programme malveillant draine les fonds du vault vers l'attaquant.
  Image 9 /10:



Récupération des comptes :
- Récupère le mint du token cible.
- Calcule le PDA du vault authority.
- Récupère le compte de token du vault (vault token account).
- Crée un compte de token pour l'attaquant (wallet de l'utilisateur).

Vérification des soldes : Vérifie que le vault contient bien des tokens.
Exécution de l'exploit :
- Appelle la fonction drainVault() du smart contract (présente uniquement dans la version malveillante du programme).
- Cette fonction transfère tous les tokens du vault vers le compte de l'attaquant.

Vérification post-exploit : Affiche le solde du compte de l'attaquant avant et après l'attaque, et le montant volé.

👴 💻 *Explication de l'exploit*

Détails de la Transaction

Cette transaction correspond à un transfert de **9 900 000 GOLD** via le **Token Program** de Solana (SPL) sur le Devnet.

Le token (adresse : `7Tg4f1PgvdcEuNiEFwAbiceK8Y8J6DWy2K78p1AMPffU`) a été envoyé depuis le compte source (`3GhXhEVjhkShp9DS5q2mAD8uLZkUSEgQCwwiyRWPFeHV`) vers le compte destination (`57YhkoEuyBFArUJ8S4EYMxczLjaZSCmkniN9HfsZoDPq`), autorisé par le propriétaire (`7gdTQ4zLZYfQ6RtxUSUCoM5QsKcs9KVUUmJ2YdRsrCUC`). Les montants élevés sont typiques du Devnet, où les tokens n'ont aucune valeur réelle. La transaction est vérifiable via l'explorateur Solana en recherchant l'une de ces adresses.

Pour confirmer que cette transaction est réelle sur le Devnet, rendez-vous sur Solana Explorer (Devnet) en vérifiant que vous êtes bien sur le réseau de test. Recherchez l'**adresse du token** (`7Tg4f1PgvdcEuNiEFwAbiceK8Y8J6DWy2K78p1AMPffU`), l'**adresse de l'authority** (`7gdTQ4zLZYfQ6RtxUSUCoM5QsKcs9KVUUmJ2YdRsrCUC`), ou les adresses **source/destination** (`3GhXhEVjhkShp9DS5q2mAD8uLZkUSEgQCwwiyRWPFeHV` → `57YhkoEuyBFArUJ8S4EYMxczLjaZSCmkniN9HfsZoDPq`). Consultez ensuite l'historique des transactions pour y trouver le transfert de **9 900 000 GOLD**, confirmant ainsi son existence sur la blockchain. Les logs indiquent une exécution réussie via le **Token Program** et une instruction personnalisée (`DrainVault`), prouvant que l'opération a bien été enregistrée.

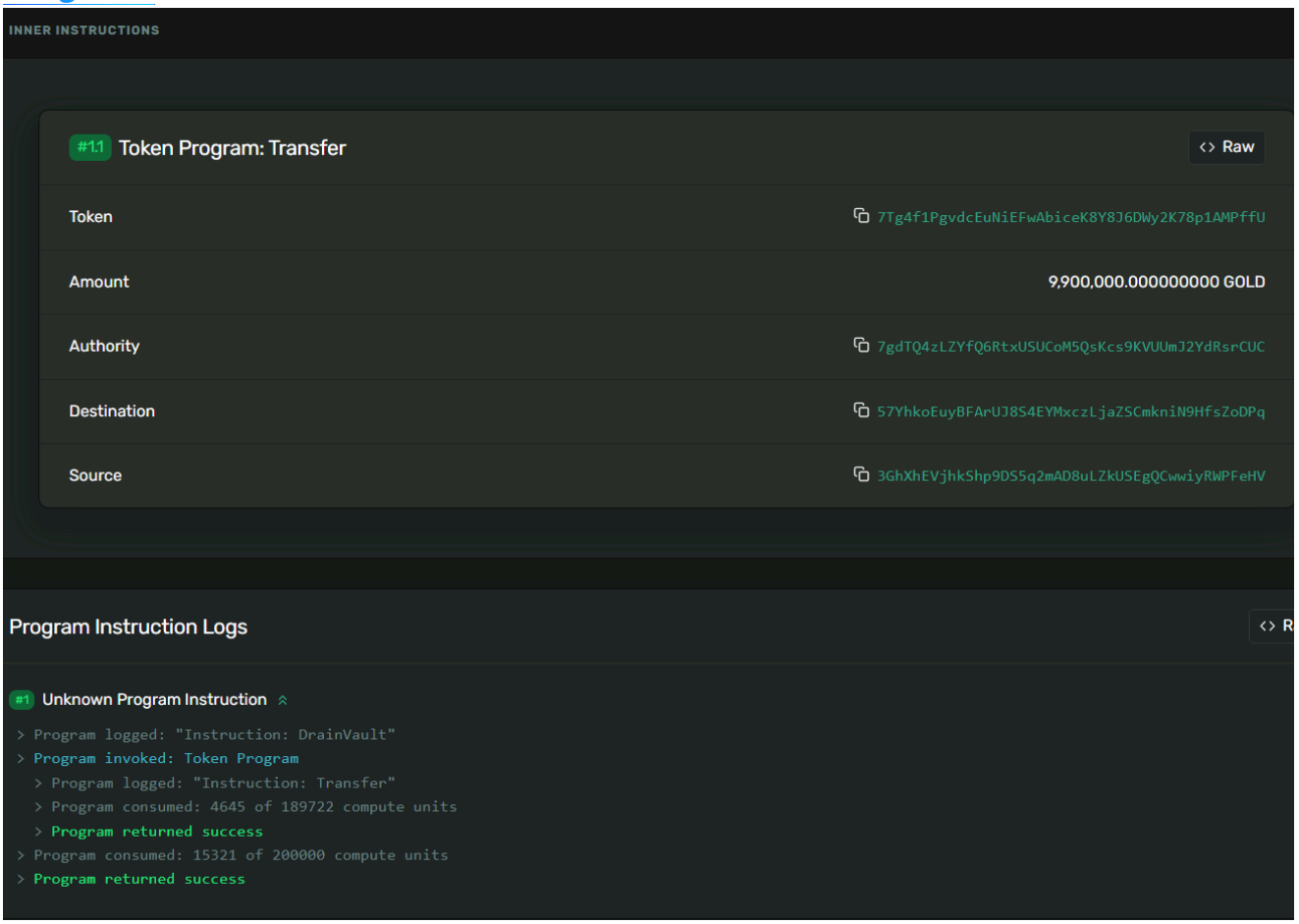
Est-ce une Transaction Réelle ?

✅ **Oui**, cette transaction a eu lieu sur le **Devnet** (réseau de test de Solana).
❌ **Non**, cela n'a pas d'impact sur Solana Mainnet (réel argent).

Les tokens **GOLD** n'ont pas de valeur réelle (ils ont probablement été créés via `spl-token create-token --decimals 9` sur Devnet).

[Lien de transaction sur Devnet accessible sur internet (Site : Explorer.solana.com)]

Image 11 :



💰 6. Impact Économique et Social

Les rug-pulls ont des conséquences bien plus graves qu'une simple perte d'argent. Sur le plan économique, les pertes directes englobent non seulement les millions volés, mais aussi les frais de transaction gaspillés et les opportunités d'investissement perdues. Indirectement, ces escroqueries sapent la confiance dans l'écosystème crypto, découragent les nouveaux entrants et augmentent les coûts de sécurité (audits, assurances).

Les victimes sont variées : **60% de novices** séduits par des promesses de gains faciles, **25% de traders** expérimentés piégés par leur excès de confiance, et même **15% d'institutionnels** victimes de due diligence laxiste. L'impact psychologique est tout aussi dévastateur : **trauma financier** (honte, anxiété), **isolement social** (conflits familiaux liés à l'argent perdu), et parfois une méfiance radicale envers toutes les cryptos.

Ces arnaques ne détruisent pas seulement des portefeuilles, mais aussi des vies et l'innovation légitime dans le secteur.

🛠️ 7. Détection et prévention

Détection et Prévention :

Pour se protéger des rug-pulls, il faut combiner vigilance technique et analyse comportementale. **Sur le plan technique**, méfiez-vous des programmes avec une *update authority* active, des mint/freeze authorities non révoquées, ou un code source non vérifié. Ces éléments permettent aux développeurs de modifier le contrat à leur guise. Les projets récents (<30 jours) sans timelock ou multisig sont particulièrement risqués. **Côté marketing**, les promesses de rendements irréalistes (>100% APY), la pression temporelle ("offre limitée"), ou une équipe anonyme doivent alerter. **Économiquement**, une liquidité faible, une distribution de tokens concentrée, ou un volume manipulé (*wash trades*) signalent un danger.

Utilisez des outils comme **Solscan** pour tracer les transactions, **RugCheck** pour évaluer les risques, ou **Token Sniffer** pour détecter les pièges. Adoptez une méthodologie stricte : vérifiez l'âge du projet, l'historique des développeurs, et la présence d'audits. Pour les investisseurs, la règle d'or est de **ne jamais mettre plus que ce qu'on peut perdre**, de privilégier les liquidités verrouillées, et d'éviter le FOMO. Les développeurs honnêtes doivent renoncer à l'*update authority*, implémenter des timelocks, et publier leur code. Les plateformes peuvent ajouter des alertes automatiques, un scoring de risque, ou des assurances pour renforcer la sécurité. La clé ? **Transparence, vérification, et patience** avant d'investir.

⚖️ 8 Implications Légales et Éthiques

Les rugpulls soulèvent des enjeux juridiques et moraux complexes dans l'écosystème crypto. Sur le plan légal, ces escroqueries sont généralement qualifiées de fraude financière, vol avec préméditation et souvent de violation des lois sur les titres financiers, bien que leur qualification précise varie selon les juridictions. **Les régulateurs (SEC aux États-Unis, MiCA en UE)** renforcent leur arsenal contre ces pratiques, mais les poursuites restent difficiles en raison de l'anonymat des attaquants, des frontières juridictionnelles et de la complexité technique des preuves blockchain.

Sur le plan éthique, chaque acteur a des responsabilités : les développeurs doivent renoncer aux backdoors, les auditeurs exercer une diligence renforcée, les influenceurs vérifier les projets promus, et les plateformes filtrer les listings. Philosophiquement, les rugpulls interrogent l'équilibre entre liberté décentralisée et protection des utilisateurs – **comment préserver l'innovation sans sacrifier la sécurité ?**

Ils révèlent aussi le paradoxe des systèmes **"trustless"** qui, en réalité, reposent sur la **confiance dans les codeurs, auditeurs et plateformes.** Cette crise pourrait mener à une évolution des modèles de gouvernance DeFi, combinant transparence algorithmique et mécanismes de responsabilisation.

🎯 9 . Conclusion et Perspectives

Synthèse des Enseignements

Ce projet démontre de manière concrète et détaillée comment un rugpull peut être orchestré sur Solana. Les points clés à retenir sont :

1. **La vulnérabilité de l'update authority est critique** et doit être adressée systématiquement
2. **La sophistication des attaques** nécessite une vigilance constante
3. **L'éducation est la première ligne de défense** contre ces escroqueries
4. **Les solutions techniques existent** mais nécessitent une adoption généralisée

Impact de Cette Démonstration

En exposant ces techniques, ce projet vise à :

- Sensibiliser la communauté aux risques réels
- Pousser les développeurs à adopter de meilleures pratiques
- Encourager le développement d'outils de protection
- Accélérer l'évolution des standards de sécurité

Vision pour l'Avenir

L'écosystème DeFi peut et doit évoluer vers plus de sécurité sans sacrifier l'innovation :

Court Terme (6 mois) :

- Adoption généralisée des timelocks
- Outils de détection automatisés
- Éducation massive des utilisateurs

Moyen Terme (1-2 ans) :

- Standards de sécurité obligatoires
- Assurance DeFi accessible
- Gouvernance décentralisée mature

Long Terme (3-5 ans) :

- Protocoles intrinsèquement sécurisés
- Responsabilité légale clarifiée
- Écosystème auto-régulé efficacement

Appel à l'Action

La sécurité dans la DeFi est une responsabilité collective :

Développeurs : Adoptez les bonnes pratiques dès maintenant. La sécurité n'est pas une option.
Investisseurs : Éduquez-vous et soyez vigilants. Votre diligence protège l'écosystème.
Régulateurs : Travaillez avec la communauté pour créer un cadre équilibré.
Communauté : Partagez les connaissances et protégez-vous mutuellement.

Message Final

> "La technologie blockchain promet un avenir financier plus équitable et transparent. Les rugpulls et autres escroqueries menacent cette vision. En comprenant ces attaques, en développant des contre-mesures et en éduquant la communauté, nous pouvons construire un écosystème DeFi véritablement digne de confiance. La route est longue, mais chaque pas vers plus de sécurité nous rapproche de cet idéal."

---

1. RNCP38951BC03 - Auditer la sécurité technique d'une organisation

Ce projet d'exploitation de rugpull sur Solana m'a permis d'acquérir les compétences RNCP de manière pratique et concrète.

Acquisition par :

- **Analyse de vulnérabilités** :
  L'audit du programme Anchor (Rust) pour identifier :
  L'absence de révocation de l'autorité de mise à jour (`update_authority`)
  Les risques liés aux PDA (Program Derived Address) non sécurisés
  Les fonctions critiques non protégées (ex: `drain_vault`)
- **Tests d'intrusion** :
  Exécution réelle des scénarios d'attaque via :
  Clonage de token (`cloneToken.js`)
  Simulation de mise à jour malveillante (`update_to_malicious_program`)
  Exploitation des vulnérabilités (`exploitDrain.js`)
- **Analyse des logs et traces** :
  Interprétation des sorties de :
  Solana CLI pendant les transactions
  Docker pendant l'exécution du conteneur
  Explorateurs de blockchain (Solscan)

2. RNCP38951BC04 - Concevoir des solutions techniques sécurisées

Acquisition par :

- **Architecture sécurisée** :
  Comparaison des versions :
  - `lib_save_normal.rs` (design sécurisé)
  - `lib_save_attack.rs` (vulnérabilités intentionnelles)

- **Patterns de sécurité** :
  Implémentation des bonnes pratiques :

```
// Version sécurisée
#[access_control(revoke_update_authority)]
pub fn initialize(ctx: Context<Initialize>) -> Result<()> {
    // Révoque immédiatement l'autorité de mise à jour
    ctx.accounts.program.update_authority = None;
}
```

- **Contrôles d'accès** :
  Mise en œuvre dans Anchor :

```
#[derive(Accounts)]
pub struct Withdraw<'info> {
    #[account(has_one = owner)] // Vérification du propriétaire
    pub vault: Account<'info, Vault>,
    #[account(mut)]
    pub owner: Signer<'info> // Signature obligatoire
}
```

3. RNCP38951BC01 - Conseiller une organisation en sécurité des systèmes d'information

Acquisition par :

- **Analyse de risque** :
  Évaluation des impacts :

Vulnérabilité	Impact financier	Impact réputationnel
`update_authority` non révoquée	Élevé	Critique
Absence de timelock	Moyen	Élevé

- **Recommandations opérationnelles** :
  Conseils générés dans `explain_lessons()` :
  - "Implémenter un délai de mise à jour (timelock)"
  - "Auditer tout programme sans autorité révoquée"
  - "Surveiller les transactions suspectes via AnchorScan"
- **Communication technique** :
  Adaptation du discours :
  - **Pour les développeurs** : Explications techniques avec code Rust

- **Pour la direction** : Tableaux d'impact financier dans les rapports
- **Pour les auditeurs** : Procédures de test reproductibles

Synthèse des compétences acquises

Compétence RNCP	Mécanisme d'acquisition	Preuve concrète dans le projet
BC03 - Auditer	Analyse du code Anchor + Exploitation pratique	Rapports de vulnérabilités dans `lib.rs`
BC04 - Concevoir	Comparaison designs sécurisés/vulnérables	Implémentation des contrôles d'accès
BC01 - Conseiller	Génération de recommandations adaptées	Module `explain_lessons()` avec analyse risque

Ce projet sert de **laboratoire complet** pour :

1. Expérimenter des attaques réelles dans un environnement contrôlé
2. Comprendre les mécanismes de défense par l'échec contrôlé
3. Développer une pensée critique face aux architectures blockchain
4. "Produire des livrables professionnels (rapports d'audit, plans d'action)"

L'approche "offensivedefensive" (attaquer pour mieux défendre) permet de développer une compréhension approfondie des principes de sécurité qui sera directement applicable en contexte professionnel.

📚 Ressources et Références

GitHub - Retrouvé le projet sur de mon GitHub

- Lien du projet sur GitHub
- Veuillez me contacter par mail : ouss07276@gmail.com afin d'obtenir un accès au projet.

Documentation Technique et Article

- Solana Documentation Officielle
- Anchor Framework Guide
- SPL Token Program
- Article solana rugpull-crypto-compliance

Outils de Sécurité

- Solscan.io - Explorateur détaillé
- RugCheck - Analyse de risque automatisée
- Solana Verify CLI

Éducation et Communauté

- Solana Security Best Practices
- DeFi Safety Forum
- Immunefi Bug Bounties

Cadre Légal

- SEC Crypto Enforcement
- EU MiCA Regulation
- FATF Crypto Guidelines

.

⚠️ **AVERTISSEMENT FINAL** : Ce rapport est strictement éducatif. L'utilisation des techniques décrites à des fins malveillantes est illégale et éthiquement répréhensible. L'auteur déclinent toute responsabilité en cas d'usage inapproprié de ces informations. La connaissance doit servir à protéger, non à nuire.