Mini_Rapport

SafeClub

1 description du contrat :

SafeClub est un contrat qui gère un trésor collectif. Les membres peuvent proposer des dépenses, voter pour les accepter ou les refuser, et un trésorier peut exécuter les propositions validées.

L'objectif de ce contrat :

- Centraliser les fonds d'un groupe de membres.
- Permettre aux membres de proposer des dépenses et de voter pour leur acceptation.
- Assurer que seules les propositions validées par une majorité peuvent être exécutées par le trésorier.
- Sécuriser le processus grâce à des contrôles d'accès et une prévention contre certaines attaques comme la réentrance.

Les acteurs principaux de ce contrat sont :

- **owner :**
  - Gérer les membres du club (Ajouter/supprimer).
  - Choisir le trésorier.
- **Membre :**
  - Réception d'ETH (vault) .
  - Création de propositions de dépenses (montant, destinataire, description, deadline).
  - Voter (pour/contre).
- **Trésorier :**
  - exécution sécurisée d'une proposition acceptée (vérifications + transfert).

2 Modèles des menaces :

2.1 Reentrancy :

La réentrance se produit lorsqu'un contrat appelle un autre contrat externe (par exemple pour transférer de l'Ether), avant de mettre à jour son état interne.

- Un contrat malveillant peut rappeler la fonction avant que l'état ne soit modifié.
- Cela permet d'exécuter plusieurs fois la logique, comme retirer des fonds plusieurs fois.

2.2 Débordement d'Entiers (Integer Overflow/Underflow) :

Un débordement d'entier (overflow) se produit lorsqu'une variable entière (`uint`, `int`) dépasse sa valeur maximale possible pour son type.

Un sous-débordement (underflow) se produit lorsqu'elle descend en dessous de sa valeur minimale.

Exemple pour un `uint8` (entier non signé sur 8 bits) :

- Valeurs possibles : 0 à 255
- Overflow : `255 + 1 → 0`
- Underflow : `0 - 1 → 255`

Sans protection, cela peut provoquer des comportements inattendus ou des vulnérabilités.

2.3 Vulnérabilités de Contrôle d'Accès :

Une vulnérabilité de contrôle d'accès (Access Control Vulnerability) se produit lorsqu'un smart contract autorise par erreur un utilisateur non autorisé à exécuter une fonction critique ou à modifier des variables sensibles.

Ces fonctions critiques peuvent inclure :

- la modification d'un rôle administratif,
- l'exécution de transferts d'Ether ou de tokens,
- l'ajout ou la suppression de membres,
- la validation ou l'exécution de décisions dans un système de gouvernance.

Si un attaquant peut contourner ces restrictions, il peut prendre le contrôle du contrat, voler des fonds, ou altérer la gouvernance, ce qui constitue un risque majeur pour tout contrat financier ou collectif

2.4 Manipulation d'Oracle de Prix :

La manipulation d’oracle de prix est une vulnérabilité qui survient lorsqu’un smart contract fait confiance à une source externe de prix (oracle) sans mécanisme de validation ou de protection. Un attaquant peut alors falsifier ou influencer le prix utilisé par le contrat, ce qui peut entraîner des décisions financières incorrectes.

Les oracles sont nécessaires car les smart contracts ne peuvent pas accéder directement aux données externes (prix des tokens, taux de change, etc.). Cependant, un oracle mal conçu devient un point de défaillance critique.

3 vulnérabilités considérées et contre-mesures :

3.1 Contre Reentrancy :

Pour protéger contre reentrancy, nous utilisons 2 mécanismes :

- Utilisation de la bibliothèque OpenZeppelin, en particulier de `ReentrancyGuard`, pour verrouiller l'exécution d'une fonction pendant qu'elle est en cours d'exécution.
- Mettre à jour l'état avant d'interagir avec des contrats externes et effectuer les validations après avoir mis à jour l'état pour les propositions exécutées et le fonction deposit d’Ethereum pour le balance de trésorier.

3.2 Contre Débordement d'Entiers :

Pour se protéger contre les débordements d’entiers, nous utilisons une version de Solidity supérieure ou égale à 0.8.0, qui effectue automatiquement des vérifications sur les opérations arithmétiques et annule la transaction en cas de débordement ou de sous-débordement.

3.3 Contre les Contrôles d'Accès :

Pour se protéger contre les attaques visant à prendre le contrôle des fonctions du contrat, nous utilisons des mécanismes de contrôle d’accès basés sur des modificateurs de fonctions. Ces modificateurs permettent de restreindre l’exécution de certaines fonctions uniquement à des rôles spécifiques. Dans notre contrat, trois modificateurs principaux sont utilisés :

- **`onlyOwner`** : ce modificateur autorise uniquement le propriétaire du contrat à exécuter certaines fonctions sensibles, telles que la gestion des membres et la désignation du trésorier.
- **`onlyMember`** : ce modificateur limite l'accès aux fonctions réservées aux membres du club, comme la création des propositions et le vote.
- **`onlyTresorier`** : ce modificateur autorise exclusivement le trésorier à exécuter la fonction de validation des propositions acceptées et à gérer le solde du trésor du club.

3.4 Autres contres-mesures :

Pour cette partie, plusieurs conditions de sécurité sont mises en place afin de garantir le bon fonctionnement du contrat. Des instructions **require** sont utilisées pour imposer des règles strictes. Par exemple, chaque membre ne peut voter qu'une seule fois pour une proposition, l'exécution d'une proposition ne peut avoir lieu qu'après l'expiration du délai , uniquement si la majorité des votes est favorable, et seulement si le solde du trésor est suffisant pour couvrir le montant à transférer. De plus , nous utilisons les événements pour faciliter de suivre les actions dans le contract.

5. Résultats d'analyse statique :

Warning 1: naming-convention

Parameter SafeClub.addMember(address).Member (contracts/SafeClub.sol#82) is not in mixedCase Pos: 82:23:

Réponse/Correction:
Nous avons renommé toutes les variables dont le nom contenait des tirets, en utilisant des noms sans tirets afin de respecter la syntaxe et les conventions de Solidity.

Warning 2: cache-array-length

Loop condition i < members.length (contracts/SafeClub.sol#93) should use cached array length instead of referencing `length` member of the storage array.

Pos: 93:28:

Réponse/Correction:
Nous stockons la longueur de table dans une variable puis nous utilisons dans une boucle.

Warning 3 : reentrancy-eth

Reentrancy in SafeClub.executeProposal(uint256) (contracts/SafeClub.sol#172-183): External calls: - (sent,None) = proposals[proposalId].recipient.call{value: proposals[proposalId].amount}() (contracts/SafeClub.sol#179) State variables written after the call(s): - tresorierBalance -= proposals[proposalId].amount (contracts/SafeClub.sol#181) SafeClub.tresorierBalance (contracts/SafeClub.sol#9) can be used in cross function reentrancies: - SafeClub.deposit(uint256) (contracts/SafeClub.sol#106-110) - SafeClub.tresorierBalance (contracts/SafeClub.sol#9)Pos: 172:4:

Réponse/Correction:

La fonction executeProposal est protégée par le modificateur `nonReentrant` fourni par ReentrancyGuard. Ce mécanisme empêche toute réentrance croisée, même si des variables sont modifiées après l'appel externe.
Ainsi, le risque de réentrance est correctement atténué, et la sécurité des fonds est assurée.

Smart Contract SafeClub

1. Présentation générale

Le contrat SafeClub est un smart contract Ethereum permettant la gestion collective de fonds au sein d'un groupe de membres. Il repose sur un mécanisme de propositions soumises au vote, avec une exécution contrôlée par un trésorier. Le contrat intègre des mécanismes de sécurité tels que le contrôle d'accès et la protection contre les attaques par réentrance.

2. Structures de données

Le contrat utilise plusieurs structures de données afin de garantir la transparence et l'immutabilité des informations:

• owner : adresse du créateur du contrat, responsable de l'administration.

• tresorier : adresse autorisée à exécuter les propositions acceptées.

• members : tableau contenant les adresses des membres.

• isMember : mapping permettant de vérifier si une adresse appartient aux membres.

• hasVoted : mapping permettant d'empêcher un membre de voter plusieurs fois.

• proposals : mapping contenant toutes les propositions

3. Structure Proposal

Chaque proposition contient un identifiant, l'adresse du proposeur, le montant demandé, le bénéficiaire, une description, une date limite de vote, le nombre de votes pour et contre, ainsi qu'un indicateur d'exécution.

4. les contrôles d'accès

- onlyOwner : Réservé au propriétaire du contrat
- onlyTresorier : Réservé au trésorier
- onlyMember : Réservé aux membres enregistrés

4. Fonctions principales

Les fonctions principales permettent la gestion des membres, la création de propositions, le vote et l'exécution des décisions.

• setTresorier() : définit le trésorier du contrat.

• addMember() / removeMember() : gestion des membres.

• deposit() : dépôt de fonds par les membres.

• createProposal(): création d'une proposition.

• voter() : vote pour ou contre une proposition.

• executeProposal() : exécution d'une proposition acceptée.

5. Règle de décision

Une proposition est acceptée si le nombre de votes pour est strictement supérieur au nombre de votes contre. Le vote doit être terminé, la proposition non exécutée, et le solde du trésor doit être suffisant. En cas d’égalité, la proposition est rejetée.

6. Sécurité

Le contrat SafeClub utilise plusieurs mécanismes pour protéger les fonds et les votes. ReentrancyGuard empêche les attaques par réentrance lors des transferts d’Ether. Les modificateurs onlyOwner, onlyTresorier et onlyMember assurent que seules les personnes autorisées peuvent exécuter certaines fonctions. Le mapping hasVoted empêche les votes multiples et des vérifications internes garantissent la validité des adresses, la disponibilité des fonds et le respect des délais. Enfin, les événements enregistrent toutes les actions importantes.