Détection des vols d'essences dans les contrats intelligents sur la plateforme Ethereum

par

Hong Huynh

Essai présenté au CeFTI

en vue de l'obtention du grade de maître en technologies de l'information

(maîtrise en génie logiciel incluant un cheminement de type cours en technologies de l'information)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Longueuil, Québec, Canada, décembre 2019

Sommaire

L'architecture des chaînes de blocs possède plusieurs caractéristiques particulières. Celle-ci est basée sur un réseau distribué, une méthode de consensus pour synchroniser les informations entre les membres, une cryptographie et un registre de données quasi immuable. En 2015, l'arrivée d'Ethereum a permis le déploiement de « contrats intelligents », c'est-à-dire des applications sur les chaînes de blocs.

Le développement de contrats intelligents requiert la compréhension de plusieurs nouveaux concepts. En effet, une fois déployées, ces applications ne sont pratiquement plus modifiables puisqu'elles résident dans un registre de données immuables. De plus, chaque invocation nécessite un coût calculé en essences qui doit être acquitté par l'appelant. Un contrat intelligent frauduleux peut se servir de plusieurs approches pour soutirer un maximum d'essences aux utilisateurs avec peu d'effort. Ce comportement est défini comme étant un vol d'essences.

Dans cet essai, l'exécution symbolique a été utilisée pour détecter les vols d'essences dans les contrats intelligents de la plateforme Ethereum. Cette méthode formelle a déjà été exploitée avec succès par d'autres chercheurs pour déceler plusieurs failles dans les contrats intelligents [1] [2] [3]. L'hypothèse de l'essai stipule que cette technique est en mesure de détecter les vols d'essences avec une efficacité d'au moins 80 %.

Un nouvel outil basé sur l'exécution symbolique, nommé Sherlock [4], a été développé et a analysé un échantillonnage de 40 contrats intelligents. Sherlock est apte à déceler les vols d'essences avec une efficacité allant jusqu'à 82,5 %. L'hypothèse de départ est de ce fait validée.

Pour atteindre un haut taux d'efficacité, Sherlock nécessite toutefois un temps d'analyse élevé de plusieurs heures. De plus, l'échantillonnage effectué comprend des contrats intelligents déployés au plus tôt un an avant la publication de l'essai. Cependant, la précision de l'outil peut être augmentée en améliorant le module d'estimation d'essences et aussi en tenant compte des appels effectués entre contrats intelligents.

Remerciements

Cet essai est le résultat d'un long parcours durant lequel j'ai eu la chance de vivre de multiples expériences enrichissantes. Je suis alors très reconnaissant envers toutes les personnes ayant contribuées de proche ou de loin à rendre ce cheminement possible.

J'aimerais d'abord remercier tout le personnel du CeFTI en commençant par M. Cardinal qui m'a permis de continuer mes études dans un programme des plus intéressants. Cet essai ne serait pas possible sans l'aide de M. Échelard et de Mme Legault qui m'ont transmis toutes les connaissances et notions nécessaires à la rédaction de l'essai. Évidemment, j'aimerais remercier mon directeur de recherche, M. Bévo Wandji qui a toujours été disponible et pour tout le temps consacré à la lecture et à l'amélioration de l'essai. Ce parcours a grandement élargi mes connaissances et j'aimerais alors remercier tous les chargés de cours du CEFTI pour leur dévouement à l'enseignement.

J'aimerais également remercier ma famille qui m'a toujours encouragé du début jusqu'à la fin et plus particulièrement ma femme qui m'a convaincu de poursuivre mes études et soutenu durant tout le processus.

Table des matières

Liste des tableaux

Liste des figures

Glossaire

Bitcoin	Première cryptomonnaie introduite en 2009 par Satoshi Nakamoto.
Chaînes de blocs	Technologie basée sur une architecture décentralisée, un registre de données, des mineurs, une méthode de consensus et la cryptographie permettant le traitement de transactions numériques à travers un réseau distribué.
Contrats intelligents	Programmes distribués déployés sur les chaînes de blocs. Ceux-ci gèrent le transfert de biens numériques.
Cryptographie	Discipline assurant la confidentialité, l'intégrité ainsi que l'authenticité des messages. Plusieurs techniques de cryptographie sont mises en usage dans les chaînes de blocs, notamment pour signer numériquement les transactions et pour préserver l'intégrité des données.
Cryptomonnaie	Monnaie virtuelle basée sur la technologie des chaînes de blocs.
Essence	Unité de mesure pour le coût d'invocation d'un contrat intelligent.
Exécution symbolique	Méthode formelle permettant la découverte et l'exploration des chemins d'exécution d'un programme informatique donné à l'aide de variables symboliques.
Ethereum	Plateforme basée sur les chaînes de blocs permettant le développement et l'exécution de contrats intelligents.
Méthode de consensus	Protocole permettant aux membres d'un réseau distribué de s'entendre sur la validité d'une transaction.

Méthode formelle	Technique permettant l'analyse rigoureuse de programmes informatiques à l'aide de formules et d'outils mathématiques afin de valider le respect des spécifications définies.
Mineur	Membre du réseau distribué dans l'architecture des chaînes de blocs.
Registre de données	Base de données distribuée en usage dans l'architecture des chaînes de blocs. Celle-ci est répliquée à travers tous les mineurs du réseau et est pratiquement immuable.
Solidity	Langage de programmation orienté objet pour le développement de contrats intelligents.
Transaction	Transfert de cryptomonnaies ou de biens numériques d'un utilisateur à un autre sauvegardé dans le registre de données.

Liste des sigles, des symboles et des acronymes

MVE	Machine virtuelle d'Ethereum (EVM en anglais)
DAO	*Decentralized Autonomous Organisation* (nom de l'application)
WSL	*Windows Subsystem for Linux*
PR	Profondeur de récursion

Introduction

En constante évolution, le domaine des chaînes de blocs est le sujet de multiples recherches [5]. Contrairement à l'infonuagique, cette technologie repose sur une architecture sans entité centrale. De plus, à la différence des bases de données traditionnelles, les transactions effectuées ne peuvent pratiquement plus être modifiées. En effet, tout changement doit être accepté par une majorité des nœuds du réseau par l'entremise d'une méthode de consensus.

Dévoilés en 2009 par Satoshi Nakamoto [6], les Bitcoins représentent l'utilisation la plus connue des chaînes de blocs. L'attrait d'une monnaie virtuelle non contrôlée par une entité centrale tel un gouvernement a fait augmenter drastiquement sa valeur [7]. Depuis, plusieurs autres cryptomonnaies ont été créées ; 2138 ont été comptabilisés [8] à ce jour en avril 2019.

En 2015, l'introduction d'Ethereum a amené une innovation majeure dans le domaine des chaînes de blocs. Cette plateforme a permis le développement ainsi que le déploiement d'applications sur les chaînes de blocs. Ces programmes distribués sont communément appelés « contrats intelligents ».

Les contrats intelligents ont rendu possible les échanges et transferts de tous biens numériques, alors qu'auparavant, seules les transactions monétaires étaient réalisables. L'invocation de ceux-ci a toutefois un coût, calculé en « essence » [9], qui doit être acquitté par l'utilisateur du service. Cependant, il est actuellement impossible de déterminer ce coût à l'avance. Les utilisateurs doivent préciser le montant maximal qu'ils sont prêts à dépenser. Un contrat intelligent frauduleux peut exploiter plusieurs techniques pour soutirer le maximum d'essences avec un minimum d'effort.

Cet essai a pour but de déceler les contrats intelligents qui ont des comportements frauduleux à l'aide d'une méthode formelle éprouvée : l'exécution symbolique.

Pour sécuriser les contrats intelligents, de plus en plus, les chercheurs ont recours aux méthodes formelles [10]. En effet, une fois déployées, ces applications distribuées sont difficilement modifiables et se retrouvent souvent dans la mire des pirates informatiques. Par

exemple, une faille dans l'application DAO a causé la perte de 50 millions de dollars aux investisseurs de ce fonds [11]. L'exécution symbolique a permis la détection de cette anomalie et de plusieurs autres avec un haut taux d'efficacité [1]. Toutefois, dans cet essai, cette méthode est appliquée pour déceler les comportements frauduleux impliquant le vol d'essences.

Dans le premier chapitre, les principaux concepts régissant le fonctionnement des chaînes de blocs sont exposés. Ainsi, cette section explique la nécessité des mineurs sur le réseau ainsi que les méthodes de consensus permettant aux membres de s'entendre sur la validité d'une transaction. De plus, les contrats intelligents ainsi que leur coût d'invocation y sont détaillés.

Le deuxième chapitre résume l'état actuel des recherches au sujet de l'utilisation des méthodes formelles dans le contexte des contrats intelligents. L'exécution symbolique représente une des techniques, mais il en existe plusieurs autres. Les avancées dans ce domaine sont multiples et ne cessent d'évoluer.

Le troisième chapitre définit la problématique, c'est-à-dire les fraudes possibles concernant le vol d'essences lors de l'invocation des contrats intelligents. De plus, une hypothèse est posée sur l'efficacité de l'exécution symbolique pour la détection de ces cas.

Le quatrième chapitre porte sur l'approche pour effectuer l'expérimentation. Plusieurs étapes sont requises. Tout d'abord, cette section définit les unités de mesure choisies pour quantifier un comportement frauduleux. Ensuite, une justification est fournie quant au choix de l'échantillonnage des contrats intelligents. Finalement, l'outil d'exécution symbolique utilisé pour cette recherche est présenté.

Le cinquième chapitre présente les données recueillies lors de l'expérimentation et leur analyse. Par la suite, une discussion est amenée sur les résultats de l'expérimentation. L'hypothèse est validée et des explications sont données pour les divergences.

En dernier lieu, l'essai émet des propositions d'amélioration et des pistes pour des recherches futures.

Chapitre 1

Mise en contexte

Les chaînes de blocs ainsi que leur principale application, la cryptomonnaie, reposent sur plusieurs concepts. Dans ce chapitre, des explications sont fournies concernant les principaux éléments régissant le fonctionnement des chaînes de blocs. Aussi, les contrats intelligents et leur coût d'invocation sont expliqués ainsi que les failles possibles. Finalement, la dernière partie clarifie la nécessité des méthodes formelles dans ce contexte.

1.1 La cryptomonnaie

L'idée de faire usage de la monnaie virtuelle remonte aux années 1980. En 1983, David Chaum a introduit la possibilité de retirer de l'argent électroniquement d'une banque pour ensuite la dépenser chez les commerçants acceptant cette méthode de paiement [12]. Un logiciel, nommé « eCash », était requis et la cryptographie assurait l'authenticité des transactions. Toutefois, cette solution n'a pas connu de succès commercial.

En 1998, Wei Dai a été le premier à proposer la création de monnaies virtuelles à travers un consensus décentralisé et en trouvant des solutions à des casse-têtes informatiques [13]. Cependant, il n'a pas émis de détails au sujet de l'implémentation de la méthode de consensus.

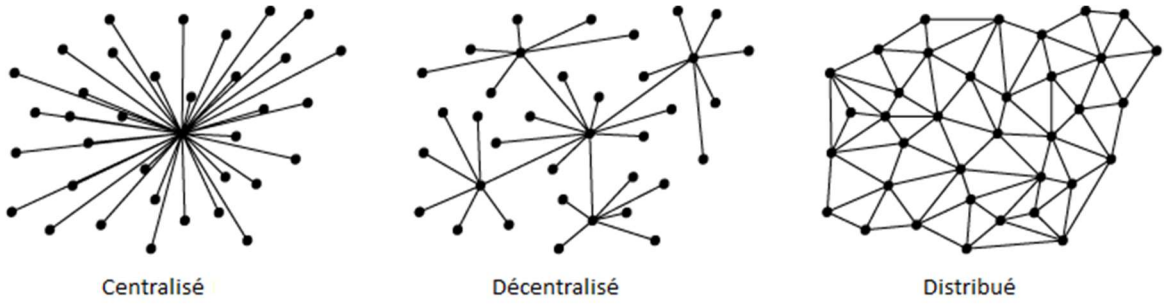
En 2009, Satoshi Nakamoto a dévoilé Bitcoin [6]. Cette cryptomonnaie révolutionnaire est basée sur une architecture pair-à-pair et distribuée sans entité centrale. Pour en assurer le fonctionnement, le créateur de cette monnaie virtuelle a dévoilé la méthode de consensus « preuve de travail ». Celle-ci permet aux pairs de communiquer entre eux et de s'entendre sur la validité des transactions.

Depuis le succès des Bitcoins, plusieurs centaines d'autres cryptomonnaies ont été lancées [8]. Celles-ci s'appuient toutes sur les chaînes de blocs qui sont basées sur les concepts suivants : une architecture distribuée, un registre de données distribuées, la cryptographie, les mineurs et une méthode de consensus.

1.1.1 Architecture distribuée

La popularité des cryptomonnaies est principalement due au fait que celles-ci ne sont pas contrôlées par une entité centrale. Pour ce faire, elles opèrent par l'entremise d'une architecture distribuée. Les membres de ce réseau, aussi appelés mineurs, sont interconnectés et se partagent toutes les informations. De plus, chacun possède un droit de vote sur la validité des opérations transigées. Une majorité doit être atteinte pour qu'une transaction soit acceptée et enregistrée. La figure 1-1 illustre les principaux types d'architecture en usage aujourd'hui. Dans les architectures centralisées et décentralisées, un ou quelques nœuds gèrent et contrôlent toutes les interactions. Ceci diffère avec l'architecture distribuée des cryptomonnaies où il y a consensus entre les membres.

Figure 1-1 : Types d'architecture



Traduction libre de : http://networkcultures.org/unlikeus/resources/articles/what-is-a-federated-network/

1.1.2 Registre de données distribuées

La technologie des chaînes de blocs repose sur un registre de données distribuées. En effet, toutes les informations sont automatiquement répliquées intégralement à travers tous les membres du réseau. L'attrait principal de ce registre est qu'il conserve un historique complet de toutes les opérations réalisées. De plus, les informations enregistrées ne sont pratiquement plus modifiables une fois acceptées.

Dans cette structure de données, les transactions sont regroupées en blocs qui sont ensuite sauvegardés les uns à la suite des autres. Ces blocs sont liés entre eux et sécurisés grâce à la cryptographie, formant ainsi une chaîne. Associée à une cryptomonnaie particulière, celle-ci

conserve alors toutes les opérations effectuées depuis sa création. La figure 1-2 illustre le contenu d'un bloc qui comprend les éléments suivants :

- Horodatage : heure et date auxquelles le bloc a été ajouté au registre;
- Données : regroupement des transactions autorisées;
- Empreinte numérique du bloc : créée en faisant le hachage des données et de l'empreinte numérique du bloc précédent;
- Empreinte numérique précédente : empreinte numérique du bloc précédent.

Figure 1-2 : Contenu du registre de données

Traduction libre de : https://medium.com/@lhartikk/a-blockchain-in-200-lines-of-code-963cc1cc0e54

Si une entité tente de modifier une donnée, cela rendra invalide l'empreinte numérique du bloc ainsi que les empreintes numériques de tous les blocs subséquents, ce qui rend cette structure difficile à trafiquer.

1.1.3 La cryptographie

Comme décrit dans la section 1.1.2, des algorithmes de hachage sont appliqués pour assurer l'intégrité des données. De plus, plusieurs autres techniques de cryptographie sont mises en œuvre pour signer numériquement les échanges de biens numériques. En effet, chaque utilisateur possède une clé privée ainsi qu'une adresse publique. Les transferts se font par l'intermédiaire des adresses publiques. La clé privée permet au propriétaire d'approuver et de signer numériquement ses opérations. Cette signature peut être validée pour confirmer le propriétaire sans toutefois révéler la clé privée.

1.1.4 Les mineurs

Les mineurs possèdent plusieurs rôles et constituent les membres du réseau dans l'architecture distribuée des chaînes de blocs. Chacun d'entre eux conserve une copie intégrale du registre de données et valide le contenu de celui-ci. En effet, les mineurs recalculent les

empreintes numériques de chaque bloc pour assurer l'intégrité des données. Pour les nouvelles transactions, lorsque celles-ci sont acceptées, ils les rassemblent en blocs et les rajoutent au registre. Puisque tout ce travail requiert du temps ainsi que de la puissance de calcul, ils reçoivent une récompense sous la forme d'une compensation pécuniaire.

1.1.5 Les méthodes de consensus

Pour que les transactions soient sauvegardées dans le registre de données, les mineurs doivent en majorité s'entendre sur leur validité. Avec Bitcoin, Satoshi Nakamoto a introduit une méthode de consensus nommée « Preuve de travail » [6] qui assure la synchronisation entre tous les intervenants et l'intégrité des données. Pour ce faire, chaque mineur tente de résoudre un casse-tête cryptographique par force brute. Le premier qui a résolu l'énigme reçoit une compensation et ajoute son bloc de transactions au registre. Il diffuse ensuite la réponse ainsi que le bloc nouvellement enregistré. Les autres mineurs sur le réseau valident alors la solution au casse-tête ainsi que les transactions dans le bloc. S'ils sont en accord, ils ajoutent également ce bloc à leur registre.

Cette méthode de consensus possède cependant une faille majeure. En effet, si une entité contrôle plus que la moitié de la puissance de calcul sur le réseau, il pourrait alors influencer les informations ajoutées au registre [14].

Plusieurs autres méthodes de consensus existent, chacun ayant leurs avantages et désavantages. Bitcoin et Ethereum font usage de la méthode « Preuve de travail ».

1.2 Les contrats intelligents

Le concept des contrats intelligents a été imaginé et détaillé par Nick Szabo en 1997 [15]. Celui-ci stipule que les contrats papier traditionnels doivent évoluer et s'adapter à la révolution numérique. Dans ce nouveau contexte, un contrat intelligent représente un programme qui exécute automatiquement des tâches basées sur des règles prédéfinies. Celui-ci permet d'entretenir des relations d'affaires par le biais de réseaux informatiques publics.

Pour la concrétisation de ce concept, plusieurs éléments sont requis :

a) Des protocoles de communication permettant d'opérer sur des réseaux publics;
b) Une infrastructure réseautique qui transfère les informations rapidement;
c) Une puissance de calcul élevée rendant possible l'accomplissement de travaux autrefois considérés comme lourds et laborieux;
d) Des techniques assurant la sécurisation des données.

Cependant, Nick Szabo n'a pas défini de méthode pour implémenter sa vision. Toutefois, ce concept est redevenu d'actualité avec l'arrivée de la cryptomonnaie. En effet, celle-ci est basée sur la technologie des chaînes de blocs qui répond aux critères nécessaires pour la réalisation des contrats intelligents.

1.3 Ethereum : une chaîne de blocs 2.0

Les premières cryptomonnaies telles que Bitcoin, Litecoin et Dogecoin, ne supportent que les échanges de monnaies virtuelles. Pour opérer, elles utilisent une architecture distribuée, un registre de données distribuées et sécurisées, un réseau public (Internet), la cryptographie, une méthode de consensus et possèdent une grande puissance de calcul. Tous ces éléments réunis rendent possible l'implémentation des contrats intelligents tels que définis par Nick Szabo [15]. En 2015, la plateforme Ethereum a été la première à concrétiser cette idée, créant ainsi une nouvelle génération de chaînes de blocs.

Les contrats intelligents sont donc des applications distribuées qui sont déployées sur les chaînes de blocs. Ceux-ci sont sauvegardés dans le registre de données et gèrent automatiquement le transfert de n'importe quels biens numériques, lorsqu'invoqués. Les mineurs sont responsables de leur exécution et reçoivent une récompense proportionnelle à la quantité de travail accompli.

1.3.1 Développement et exécution des contrats intelligents

Le développement des contrats intelligents sur Ethereum se fait principalement avec Solidity, un langage de programmation de haut niveau ayant une syntaxe semblable à JavaScript [16]. D'autres langages existent, mais ne sont pas recommandés. Par exemple, Serpent comporte

des failles de sécurité [17] et LLL est un langage de bas niveau. Peu importe l'outil de développement choisi, le code source est traduit en code intermédiaire qui sera interprété par la machine virtuelle d'Ethereum (MVE).

La MVE est un environnement d'exécution à base de pile pour les contrats intelligents. Celle-ci implémente un système Turing-complet [18], permettant des opérations avancées telles que les boucles et les fonctions récursives. Chaque mineur implémente une MVE et est responsable de l'exécution des contrats intelligents.

1.3.2 Coûts de l'invocation des contrats intelligents

Pour évaluer l'effort déployé par les mineurs, l'essence est l'unité de mesure privilégiée par Ethereum. L'exécution de chaque instruction coûte un certain nombre d'essences prédéfini à l'avance [9]. Plus un contrat intelligent nécessite d'opérations pour effectuer une tâche, plus son invocation sera coûteuse. L'invocateur a la responsabilité d'acquitter les frais.

Avant d'appeler un contrat intelligent, l'utilisateur doit spécifier la quantité maximale d'essences qu'il est prêt à dépenser pour le service. Si celle-ci est insuffisante, l'application lance une exception. La transaction est alors annulée et tous les changements sont renversés.

Le coût de l'invocation d'un contrat intelligent ne peut pas être connu d'avance [19]. Des outils d'estimation existent, mais ne sont pas précis. En effet, ceux-ci font leur évaluation en utilisant le contexte courant des chaînes de blocs. Ce contexte peut changer entre le moment de l'estimation et le moment réel de l'exécution.

1.3.3 Failles dans l'invocation des contrats intelligents

La plateforme Ethereum ne possède qu'une seule exception intitulée « quantité d'essences insuffisante ». Celle-ci est lancée dans les cas suivants :

- La quantité d'essences spécifiée par l'invocateur est insuffisante pour la tâche demandée;
- Le contrat intelligent rencontre une erreur fatale (division par zéro, atteinte de la limite maximale du nombre d'appels, indice d'un tableau hors limite, assertion non respectée, appel d'une fonction avec paramètre invalide);
- Le contrat intelligent lance délibérément l'exception.

Un contrat intelligent malhonnête peut tirer avantage des cas ci-dessus pour soutirer le plus d'essences possible aux utilisateurs avec un minimum d'effort. Par exemple, celui-ci décide de changer de comportement et lance volontairement une exception pour récolter toute l'essence de l'invocateur sans donner le service en retour. Un autre scénario possible est que celui-ci modifie la quantité d'essences nécessaire pour l'exécution en fonction de paramètres imprévisibles du contexte des chaînes de blocs comme l'empreinte digitale du bloc de transactions précédent. Puisqu'Ethereum est une chaîne de blocs publique, n'importe qui peut se joindre au réseau et déployer un contrat intelligent à condition de payer les frais de déploiement.

1.4 Les méthodes formelles

Pour sécuriser les applications critiques, le domaine du génie logiciel a recours aux méthodes formelles. Les contrats intelligents sont considérés comme critiques puisqu'ils gèrent souvent de grandes sommes d'argent qui attirent les pirates informatiques. De plus, une fois déployés, ceux-ci ne peuvent plus être modifiés, ne laissant pas le droit à l'erreur. Finalement, le développement de ces applications distribuées nécessite la maîtrise de nouveaux concepts, ce qui complique la tâche pour les nouveaux développeurs [20].

Après la faille du DAO [11], la communauté et la fondation Ethereum se sont tournées vers les méthodes formelles. Cependant, l'efficacité de leur application dans ce nouveau contexte n'a pas encore été démontrée. Les chercheurs explorent plusieurs approches possibles, allant de l'exécution symbolique à la formalisation du langage de développement ou de la machine virtuelle. L'état actuel des recherches est présenté dans le chapitre suivant.

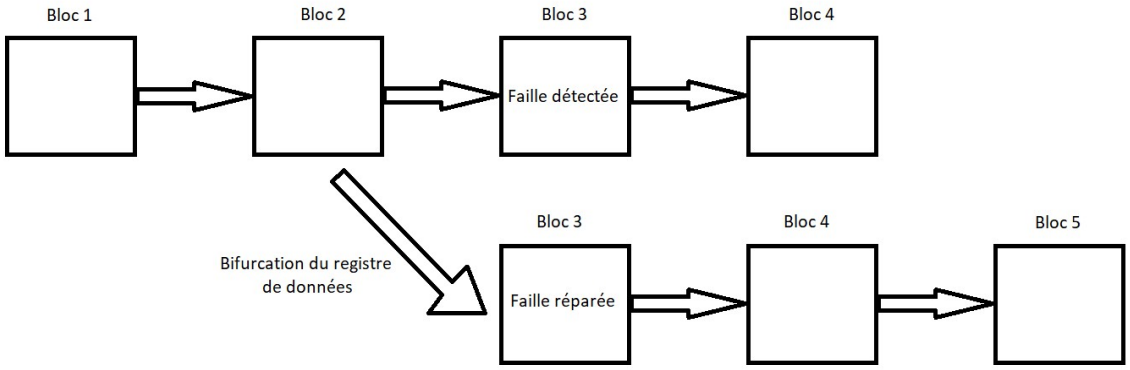
Chapitre 2

Revue de la littérature

Le concept des contrats intelligents implémentés sur les chaînes de blocs est un phénomène récent. En 2015, Ethereum est devenu la première plateforme permettant le développement et le déploiement de ces applications distribuées. Depuis, plus d'un million de contrats intelligents sont exploitables [21]. Puisque cette technologie offre de nouvelles possibilités, il existe un fort engouement pour la recherche dans ce domaine [5].

Une fois déployé, un contrat intelligent n'est pas modifiable à moins d'utiliser une méthode controversée. En effet, il est possible de revenir à un état antérieur du registre de données et de forcer une bifurcation pour annuler les changements qui ont été effectués. La figure 2-1 illustre cette situation.

Figure 2-1 : Bifurcation du registre de données



Cette démarche a été mise en œuvre pour rectifier une faille [22] et a profondément divisé la communauté d'Ethereum. Cette façon d'opérer va à l'encontre des principes des chaînes de blocs où tous les changements sont finaux. Pour éviter qu'une telle situation ne se reproduise, l'utilisation des méthodes formelles est devenue une nécessité.

Dans ce chapitre, la méthode de recherche utilisée pour la revue de littérature est décrite. Ensuite, dans la poursuite de l'objectif de déployer des contrats intelligents sans anomalies, les expérimentations des chercheurs avec diverses techniques formelles sont résumées.

2.1 Méthodologie de recherche

2.1.1 Mots-clés utilisés

Les recherches ont été effectuées principalement avec l'outil de découverte de l'Université de Sherbrooke ainsi que Google Scholar. Celles-ci se sont avérées compliquées en raison de la grande quantité de faux résultats. Le tableau 2-1 résume les termes ainsi que les critères utilisés lors des recherches pour la revue de littérature. Le nombre de résultats obtenus ainsi que la quantité d'articles pertinents retenus y sont détaillés. Souvent, une inspection du titre était suffisante pour écarter un écrit. Finalement, le tableau mentionne le nombre de résultats uniques correspondant à une combinaison de mots-clés. Les dernières recherches ne fournissaient pas de nouveaux articles reliés au sujet ce qui suppose l'atteinte d'un point de saturation.

Tableau 2-1 : Termes utilisés pour la recherche d'articles et quantité de résultats par outil

Outil de découverte — Université de Sherbrooke				
Mots-clés	Critères	Résultats	Résultats retenus	Résultats uniques
Formal Methods ET (Ethereum OU Blockchain)	Révisé par pairs Après 2015	251	5	0
Smart Contracts ET Ethereum	Révisé par pairs Après 2015	961 (20 premières pages inspectées)	17	2
Smart Contracts ET Blockchain	Révisé par pairs Après 2015	2370 (20 premières pages inspectées)	8	1
Smart Contracts ET Fraud	Révisé par pairs Après 2015	4014 (20 premières pages inspectées)	2	0
Google Scholar				
Mots-clés	Critères	Résultats	Résultats retenus	Résultats uniques
Formal Methods ET Ethereum	Après 2015 Tri par pertinence	1560 (25 premières pages inspectées)	13	5

Formal Methods ET Blockchain	Après 2015 Tri par pertinence	5390 (10 premières pages inspectées)	9	1
Smart Contracts ET Ethereum	Après 2015 Tri par pertinence	6990 (25 premières pages inspectées)	22	5
Smart Contracts ET Blockchain	Après 2015 Tri par pertinence	14400 (25 premières pages inspectées)	12	0
(Smart Contracts OU Ethereum) ET Fraud	Après 2015 Tri par pertinence	14500 (25 premières pages inspectées)	2	0

2.2 Formalisation de la machine virtuelle d'Ethereum (MVE)

Les contrats intelligents de la plateforme Ethereum sont tous traités par la MVE. Une approche pour prouver la validité et l'exactitude des exécutions consiste à formaliser la MVE pour que celle-ci soit analysable par un assistant de preuve. Ensuite, cette définition formelle peut être mise à profit pour vérifier formellement les contrats intelligents. Hirai [23] s'est servi de cette méthode et a traduit la MVE en Lem [24], un langage qui est interprétable par plusieurs assistants de preuve tels qu'Isabelle/HOL. De plus, son implémentation couvre toutes les instructions de la MVE. Il a pu par la suite vérifier formellement trois propriétés de bas niveau des contrats intelligents. Cependant, ces validations requièrent beaucoup de temps, c'est-à-dire trois heures pour un seul contrat sur une machine moderne. Aussi, la preuve formelle de seulement trois propriétés n'est pas suffisante pour assurer qu'un contrat soit sans failles.

Hildenbrandt [25] a employé la même approche que Hirai, mais cette fois en formalisant la MVE à l'aide du cadriciel K [26] et en apportant plusieurs améliorations. Son implémentation, nommée « KEVM », a passé avec succès les 40 783 cas de tests officiels pour une MVE, ce qui la rend plus complète. De plus, l'exécution de la solution de Hildenbrandt prend huit fois moins de temps que celle de Hirai, ce qui la rend utilisable dans des conditions réelles. Aussi, le cadriciel K est très flexible et comprend plusieurs outils, dont des engins d'exécution concrète et symbolique. Ceux-ci n'ont pas été mis à profit dans la recherche, mais offrent des possibilités futures. Toutefois, seulement deux propriétés des contrats intelligents ont pu être prouvées lors de l'expérimentation.

2.3 Traduction du code source et du code intermédiaire en F*

Dans la section précédente, les chercheurs se sont concentrés à examiner le code intermédiaire exécuté par la MVE. Le développement des contrats intelligents se fait généralement avec des langages de haut niveau comme Solidity [16]. Il est alors nécessaire de s'assurer qu'aucune erreur n'est introduite lors de la compilation du code source. Bhargavan [27] tente de résoudre cette problématique en proposant de traduire le code source et le code intermédiaire en F*, un langage fonctionnel employé pour la vérification. La validation s'effectue donc à deux niveaux et les résultats peuvent être comparés pour prouver que le compilateur a converti fidèlement le code source.

Cette recherche est encore exploratoire et l'analyse s'est restreinte à des cas simples. Le traducteur de Solidity à F* proposé ne supporte pas les boucles, ce qui limite l'échantillonnage possible (46 contrats sur les 396 accessibles au moment de la recherche). Similairement, l'outil pour la traduction du code intermédiaire à F* est également incomplet et n'implémente pas toutes les instructions de la MVE.

2.4 Exécution symbolique et analyse statique du code intermédiaire

2.4.1 Détection de failles

Dans l'objectif de détecter les failles majeures, Luu [1] propose d'utiliser l'exécution symbolique pour analyser le code intermédiaire des contrats intelligents. En effet, cette méthode formelle détermine tous les chemins possibles d'un programme et construit un graphe de contrôle. Ensuite, chaque bifurcation est traversée et inspectée. Luu a introduit l'outil Oyente qui accomplit ces actions et qui est en mesure de déceler quatre failles majeures, dont celle du DAO. De plus, l'inspection de chaque contrat intelligent ne prend que six minutes en moyenne sur une instance large d'Amazon EC2. L'échantillonnage de la recherche est large et comprend 19 366 contrats intelligents dont 8 833 d'entre eux ont été signalés comme ayant une des quatre failles visées. Cependant, l'outil Oyente retourne quelques faux résultats. Après avoir étudié manuellement 175 contrats signalés comme ayant un défaut, Luu a découvert que dix contrats n'avaient aucune faille. Oyente a alors un taux de mauvaise détection de 6,4 %.

Mis à part Oyente, il existe d'autres outils qui utilisent l'exécution symbolique pour analyser les contrats intelligents d'Ethereum. Par exemple, Mythril [28] a été présenté en novembre 2018 à la conférence de sécurité HITBSECCONF2018 à Amsterdam. Cet outil est développé par la firme Consensys qui a été mise en place par Joseph Lubin, un des cofondateurs d'Ethereum. Mythril est plus récent et permet de détecter plus de failles qu'Oyente [29]. Il est activement mis à jour [3] avec de nouvelles fonctionnalités, telles que l'estimation d'essences. Cependant, il retourne aussi des faux résultats, autant qu'Oyente [30].

2.4.2 Détection de pratiques de programmation inefficaces

L'invocation des contrats intelligents n'est pas gratuite et le coût est proportionnel à la quantité d'instructions exécutées. Les développeurs ont donc intérêt à écrire du code efficace. Chen [2] a identifié sept pratiques de programmation jugées inefficaces et se sert de l'exécution symbolique pour les déceler. En effet, Oyente, introduit initialement par Luu [1], a été adapté pour détecter trois de ces mauvaises pratiques. Le nouvel outil, nommé Gasper, a identifié que plus de 80 % des contrats intelligents souffraient d'au moins une de ces trois pratiques de programmation inefficaces. Cependant, l'inspection est effectuée sur le code intermédiaire alors que le développement est effectué dans Solidity. Il est alors difficile pour un développeur de cibler le lieu exact de l'erreur.

2.5 État actuel des recherches

Les méthodes formelles appliquées au domaine des contrats intelligents sont encore à l'étape de recherche. Actuellement, aucune approche ne garantit la détection ou le déploiement d'un contrat intelligent sans failles. Cependant, les travaux de Hildenbrandt [31] sont prometteurs et le chercheur a reçu du financement additionnel. Park [32] a ajouté des fonctionnalités dans l'outil KEVM de Hildenbrandt pour analyser formellement quelques contrats intelligents populaires. Hirai travaille pour la fondation Ethereum et poursuit ses démarches. Quant à Luu, il a pu continuer ses recherches [33] et a déployé une version bêta de l'outil Oyente. La compagnie Consensys poursuit le développement de Mythril et a aussi rendu disponible une interface de programmation nommée MythX. Oyente et Mythril sont intéressants puisqu'ils sont

libres de droits, modulaires et adaptables au besoin. En effet, plusieurs chercheurs ont utilisé Oyente pour détecter divers cas :

- Chen [2] : création de Gasper pour la détection de pratiques de programmation inefficaces;
- Torres and Steichen [34] : création de HoneyBadger qui permet de déceler les contrats intelligents contenant des leurres;
- Torres [35] : création d'Osiris pour trouver les erreurs arithmétiques comme la division par zéro;
- Albert [36] : création de Gastap pour déterminer la quantité maximale d'essences nécessaire lors de l'invocation d'un contrat intelligent. Cette recherche fournit seulement un seuil supérieur pour le coût en essences et ne mentionne pas de comportements de vols d'essences. Aussi, le code source n'a pas été publié.

D'autres outils d'exécution symbolique sont également disponibles :

- Manticore [37] : développée par la compagnie « Trail of Bits », cette application est libre de droits et possède une version bêta. Celle-ci met à la disposition de l'utilisateur plusieurs fonctions permettant l'analyse symbolique d'un programme. Cependant, peu d'articles réfèrent à Manticore et son efficacité reste encore à prouver;
- Securify [38] : initialement dévoilé en 2017, Securify a été financé par la fondation Ethereum. Il traduit le code intermédiaire des contrats intelligents en Datalog et permet la définition de propriétés dans ce même langage. Ces propriétés sont ensuite validées pour tous les chemins détectés grâce à l'exécution symbolique. Le code source de cette application a été rendu disponible en fin septembre 2018 [39] [40];
- MAIAN [41] : cet outil combine l'exécution symbolique et des invocations répétées pour détecter trois types de cas : les contrats intelligents suicidaires, gourmands et prodigues. Cependant, MAIAN ne prend en compte que 121 des 133 instructions de la MVE, ce qui limite les contrats intelligents analysables.

Certains outils décrits dans cette section proviennent du secteur privé. L'exécution symbolique est perçue comme une technique ayant le potentiel de renforcer les contrats intelligents dans les secteurs universitaire et privé.

Chapitre 3

Problématique

Le vol d'essences dans les contrats intelligents représente une problématique sérieuse qui est souvent ignorée. En effet, les chercheurs dans le domaine se concentrent principalement à remédier aux failles majeures et aux fraudes qui ont un impact financier élevé [14] [42] [1] [25]. Cependant, le vol d'essences permet d'usurper de petits montants répétés sur une longue période. Les conséquences à long terme sont graves puisque ce type de fraude mine la confiance des utilisateurs et empêche l'adoption des contrats intelligents du public. De plus, la perte financière est pratiquement irréversible dû à la nature des chaînes de blocs. Les utilisateurs se retrouvent donc floués et sans recours.

Pour pallier les vols d'essences, la création d'un outil pouvant détecter ce comportement indésirable est essentielle. Celui-ci permettrait aux utilisateurs de faire un choix éclairé et également de signaler les contrats intelligents malicieux. Pour atteindre cet objectif, des recherches doivent être menées. En effet, elles sont nécessaires pour définir et quantifier les caractéristiques du vol d'essences ainsi que pour déterminer la technique la plus appropriée pour déceler ce comportement.

3.1 Le coût d'utilisation des contrats intelligents

Dans les chaînes de blocs publiques, n'importe quel ordinateur peut se joindre au réseau pour faire partie du groupe des mineurs. Ceux-ci fournissent une puissance de calcul nécessaire à l'opération des contrats intelligents. Pour empêcher qu'un contrat intelligent ne monopolise toutes les ressources des mineurs, l'exécution de chaque instruction possède un coût calculé en essences. Ces frais doivent être acquittés par les utilisateurs. Ce mécanisme n'est pas parfait. En effet, le coût est connu seulement une fois que le contrat intelligent a terminé ses tâches. Avant l'invocation, l'utilisateur doit estimer le travail à accomplir et déterminer le nombre maximal d'essences qu'il est prêt à dépenser pour le service. Si l'estimation est inférieure au

coût réel, le contrat intelligent annule tous les changements effectués, mais garde quand même l'essence de l'utilisateur.

Des outils d'estimation d'essences sont disponibles, mais ne sont pas précis [43]. En effet, ils effectuent leurs calculs en fonction du contexte courant des chaînes de blocs. Ce contexte évolue continuellement et sera différent au moment de l'exécution réelle. Par exemple, un contrat intelligent frauduleux peut faire varier son coût d'invocation volontairement en fonction de plusieurs paramètres tels que l'heure ou la signature électronique du bloc précédent. Ces stratagèmes de vol d'essences ne sont pas détectables avec les outils actuels. Cependant, les méthodes formelles peuvent remédier à cette situation.

3.2 Les méthodes formelles

L'industrie du génie logiciel a recours aux méthodes formelles pour sécuriser les applications critiques [44]. Les contrats intelligents sont considérés comme étant critiques dus aux grandes sommes d'argent que ceux-ci gèrent et au caractère irréversible des chaînes de blocs. Depuis 2016, la communauté d'Ethereum fait usage des méthodes formelles pour déceler les failles potentielles [45]. Cependant, l'application de ces techniques aux contrats intelligents d'Ethereum est un phénomène récent. Les chercheurs ont exploré plusieurs approches qui sont résumées dans la prochaine section.

3.3 L'application des méthodes formelles aux contrats intelligents d'Ethereum

Les chercheurs Hildenbrandt [25] et Hirai [23] ont formalisé la machine virtuelle d'Ethereum dans le but de valider les contrats intelligents à l'aide d'un assistant de preuve. Quoique fonctionnelle, cette méthode nécessite plus d'approfondissements puisqu'elle ne permet que la validation de quelques propriétés de bas niveau.

Bhargavan [27] a tenté une autre approche dans l'objectif de prouver que le compilateur du langage Solidity n'introduit pas d'erreurs lors de la transformation en code intermédiaire. Le chercheur traduit alors à la fois le code source et le code intermédiaire en F*, un langage formel.

Les deux résultats sont ensuite comparés pour démontrer que le compilateur a bien effectué la conversion. Cependant, les outils de traduction en F* ne supportent pas toutes les instructions, ce qui limite l'application de la méthode à des contrats intelligents simples.

Luu [1] se sert de l'exécution symbolique pour vérifier le code intermédiaire des contrats intelligents. Il a introduit l'outil d'analyse Oyente qui a permis de déceler quatre failles majeures avec un haut taux d'efficacité. En effet, après analyse du code source, Oyente n'a retourné que 10 cas faux positifs sur 175 pour un taux d'erreur de 6,4 %. Cette méthode est également très flexible puisqu'elle a été réutilisée par Chen [2] pour détecter plusieurs pratiques de programmation inefficaces ainsi que par d'autres chercheurs pour déceler diverses failles. Plus récent qu'Oyente, Mythril [3] détecte plus de failles et est en constante amélioration par Consensys, une firme créée par un des cofondateurs d'Ethereum.

L'exécution symbolique a été retenue comme méthode formelle d'usage dans cet essai. En effet, la flexibilité et le haut taux d'efficacité de cette technique sont des critères essentiels pour cette recherche. Cependant, le rendement exact de l'exécution symbolique pour découvrir les comportements de vols d'essences reste à déterminer.

3.4 Efficacité de l'exécution symbolique pour la détection des vols d'essences

Selon Luu [1], l'exécution symbolique appliquée sur les contrats intelligents représente une méthode efficace pour la vérification de ceux-ci. En effet, l'outil Oyente possède un taux d'erreur bas de 6,4 % lors de la détection de quatre failles (10 mauvais résultats sur 175 contrats étudiés). De plus, la flexibilité de cet outil a été prouvée puisqu'il a été réutilisé dans d'autres contextes [2] [34] [35]. Mythril possède un taux de détection comparable à Oyente pour deux failles et un taux de faux résultats plus élevé pour une faille [30]. Il permet aussi l'ajout de nouvelles fonctionnalités par l'entremise de modules.

En fonction du cadre théorique, il est possible de déceler les comportements de contrats intelligents qui ont pour but de voler de l'essence avec l'exécution symbolique. Cependant, l'efficacité de la détection dépend de la qualité de la définition des comportements visés. De plus, une quantité faible de faux résultats dus à l'utilisation de l'exécution symbolique est

également attendue [1] [30]. En tenant compte du taux d'erreur de la méthode et de la complexité à formaliser les scénarios voulus, le taux d'efficacité prévu est défini dans la question de recherche.

3.5 La question de recherche

L'objectif de l'essai est de déterminer si l'exécution symbolique représente une méthode formelle efficace pour détecter certaines fraudes de vol d'essences dans les contrats intelligents.

La question de recherche est posée comme suit :

Est-ce que l'exécution symbolique appliquée sur les contrats intelligents de la plateforme Ethereum permet de détecter deux comportements connus dont le but est de voler de l'essence avec un taux d'efficacité d'au moins 80 % ?

Dans l'essai, les deux comportements suivants sont visés :

1) Le contrat intelligent lance une exception de façon « volontaire » ou « involontaire »;
2) Le coût en essences varie en fonction de paramètres des chaînes de blocs tels l'empreinte numérique ou l'horodatage du bloc de transactions précédent.

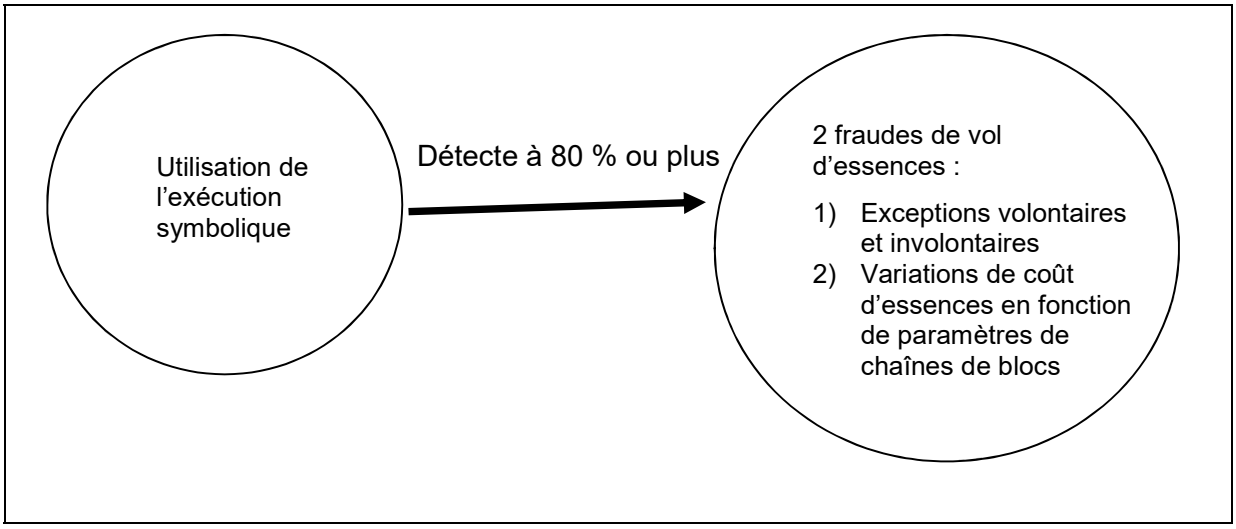
Ces deux comportements ont été choisis à la suite de notre recherche. En effet, la commande « throw » dans Solidity lance délibérément une exception qui annule tous les changements et, si le contrat intelligent est compilé avec une version de Solidity antérieure à 0.4.10, il consomme toute l'essence de l'utilisateur. L'usage de cette commande n'est pas recommandé et elle ne sera plus accessible dans le futur [16]. Les exceptions involontaires (division par zéro, indice de tableau hors limite, assertion non respectée) causent l'arrêt brusque de l'exécution et consomment également toute l'essence restante. De plus, un contrat intelligent est considéré comme frauduleux si son comportement varie en fonction de paramètres qui ne devraient pas avoir d'impact sur son fonctionnement. Par exemple, une application malsaine peut décider de doubler son coût d'invocation si l'empreinte numérique du bloc précédent est un chiffre pair.

Le taux d'efficacité de 80 % est estimé en tenant compte du taux d'erreur des outils d'exécution symbolique calculé à 6,4 % pour Oyente [1] et de la complexité à quantifier les deux comportements visés. À notre connaissance, l'exercice de définir ces comportements n'a jamais été tenté auparavant. En effet, Dika [46] a réalisé une revue de tous les outils de sécurité pour les contrats intelligents ainsi que les failles que ceux-ci peuvent détecter. Aucune mention des vols d'essences n'est présente. De plus, la revue de littérature n'a pas révélé d'éléments pouvant orienter cette activité.

3.5.1 Le schéma conceptuel et l'hypothèse

Le schéma conceptuel présenté à la figure 3-1 résume la question de recherche. Dans cette représentation, la variable prédictive est la méthode formelle choisie en fonction de la revue de littérature, c'est-à-dire l'exécution symbolique. Celle-ci est utilisée pour prédire la variable prédite, c'est-à-dire les contrats intelligents qui ont un des deux comportements de vols d'essences. Finalement, l'objectif de la recherche est de déterminer si l'efficacité de la méthode sélectionnée dans le contexte de détection de fraudes est d'au moins 80 %.

Figure 3-1 : Schéma conceptuel de la recherche



En fonction de la théorie, l'hypothèse est que oui, l'exécution symbolique permet de détecter les fraudes de vols d'essences dans les contrats intelligents avec une efficacité d'au moins 80 %.

3.6 Limites

Cet essai se limite à l'utilisation d'une seule méthode formelle, c'est-à-dire l'exécution symbolique. Le but n'est pas de comparer les différentes techniques formelles, mais de déterminer l'efficacité de l'exécution symbolique dans un contexte précis.

De plus, plusieurs chaînes de blocs supportent l'utilisation des contrats intelligents telles que Cardano [47], EOS [48] ou Tron [49]. Cependant, l'essai se concentre sur la plateforme Ethereum [50] qui possède le plus grand nombre de contrats intelligents [51].

Finalement, plusieurs failles de sécurité existent dans les contrats intelligents [52]. Cette recherche investigue seulement deux cas de vol d'essences.

3.7 Conclusion

Le vol d'essences dans les contrats intelligents est une problématique peu étudiée. Pourtant, cette fraude a des répercussions majeures sur la confiance des utilisateurs des chaînes de blocs et rend difficile l'adoption de la technologie. Cet essai effectue des démarches pour déceler les contrats intelligents frauduleux en utilisant la méthode formelle de l'exécution symbolique. La détection de ces cas apporte une contribution au domaine puisque la fondation Ethereum a reconnu que la fonction actuelle d'estimation d'essences comporte des lacunes [43].

La prochaine section décrit les étapes nécessaires pour effectuer l'expérimentation. Puisque la recherche tente de prédire l'efficacité d'une méthode, un devis de type expérimental prédictif est adopté.

Chapitre 4

Méthodologie

Le chapitre précédent a décrit la problématique et a posé la question de recherche comme suit : « Est-ce que l'exécution symbolique appliquée sur les contrats intelligents de la plateforme Ethereum permet de détecter deux comportements de vol d'essences avec une efficacité d'au moins 80 % ? » Pour répondre à cette interrogation, une recherche quantitative de type corrélationnelle prédictive est effectuée. En effet, plusieurs écrits ont déjà prouvé l'existence d'une relation d'association entre l'exécution symbolique et les contrats intelligents [8] [23] [34] [35] [30]. Cet essai tente alors de prédire l'efficacité de cette méthode formelle, lorsqu'appliquée à un contexte particulier, c'est-à-dire la détection de fraudes d'essences.

La méthodologie proposée comporte plusieurs étapes. Tout d'abord, il est nécessaire de caractériser les deux comportements frauduleux avec des mesures précises, c'est-à-dire en fonction des chemins d'exécution et de la cause de ceux-ci. Ensuite, l'outil d'exécution symbolique choisi doit être adapté pour permettre la détection des cas voulus. Pour valider le bon fonctionnement de ce nouvel instrument de mesure, des contrats intelligents frauduleux sont créés dans l'environnement test d'Ethereum et sont ensuite soumis en entrée dans l'application. Par la suite, cet outil est mis en usage pour analyser un échantillonnage des contrats intelligents réels en production. Finalement, une analyse statistique est entreprise pour déterminer l'efficacité de la méthode.

4.1 Définition des comportements frauduleux de vol d'essences

La première étape de la méthodologie consiste à définir les deux comportements de vol d'essences avec des mesures concrètes. Dans le cadre de l'expérimentation, un contrat intelligent frauduleux dont le but est de voler de l'essence est défini comme suit : tout contrat intelligent qui emprunte volontairement un chemin d'exécution plus coûteux en essences en se

fiant à des paramètres de chaînes de blocs. De plus, celui-ci est également considéré comme frauduleux s'il lance volontairement ou involontairement une exception pour terminer abruptement l'exécution et ainsi subtiliser toute l'essence de l'utilisateur.

Basé sur la définition ci-dessus, le tableau 4-1 détaille les mesures en usage dans la recherche ainsi que la raison de l'utilisation de chacune d'entre elles. De plus, celles-ci sont sélectionnées en fonction des capacités de la méthode formelle choisie, c'est-à-dire l'exécution symbolique.

Tableau 4-1 : Mesures sélectionnées pour caractériser les vols d'essences

Mesures	Explications
Nombre de chemins d'exécution	Un contrat intelligent frauduleux comporte au moins un chemin dont le coût en essences est plus élevé que nécessaire.
Quantité d'essences par chemin d'exécution	
Cause de la bifurcation de chaque chemin d'exécution	La cause de la bifurcation est essentielle pour déterminer si celle-ci est légitime ou non.
Présence de l'instruction « ASSERT_FAIL »	Cette instruction dans la MVE cause la fin de l'exécution et consomme toute l'essence.

L'objectif de l'utilisation de ces mesures est de déterminer si chaque chemin emprunté par un contrat intelligent est légitime ou dépend d'une des conditions considérées comme frauduleuses. Si un chemin illégitime existe et requiert une quantité d'essences plus importante que les chemins avoisinants, le contrat intelligent est alors catégorisé comme frauduleux.

4.2 Critères de détection par l'exécution symbolique

Dans les chaînes de blocs, le code intermédiaire des contrats intelligents est accessible, contrairement au code source. Pour cette raison, les outils d'analyse opèrent souvent sur le code intermédiaire, ce qui est le cas d'Oyente et de Mythril. Ethereum possède 139 instructions différentes [53] qui sont exécutées par la MVE. L'analyse de ces instructions est nécessaire dans le but de cueillir les mesures décrites dans la section 4.1.

4.2.1 Caractérisation d'un chemin d'exécution

Un chemin d'exécution est muni d'un début et d'une fin. Dans Ethereum, l'exécution d'un contrat intelligent débute toujours de la même façon. Les pointeurs et les espaces mémoire sont tout d'abord initialisés. Par la suite, selon le service demandé par l'utilisateur, un sélecteur de fonction appelle le code approprié. Finalement, l'exécution se termine dans les cas suivants :

- Le contrat intelligent a terminé les opérations demandées, avec ou sans retour de données;
- Les conditions d'exécution ne sont pas respectées (ex. : paramètres non adéquats);
- Une exception volontaire a été lancée (commande « throw » dans Solidity);
- Une exception involontaire est survenue (quantité d'essences insuffisantes, erreurs arithmétiques, indice de tableau invalide, assertion non respectée).

Dans le cadre de cet essai, les cas définis ci-dessus sont examinés. Le tableau 4-2 affiche les instructions de la MVE qui terminent l'exécution ainsi que la raison de l'arrêt.

Tableau 4-2 : Instructions de la MVE indiquant la fin d'un chemin d'exécution

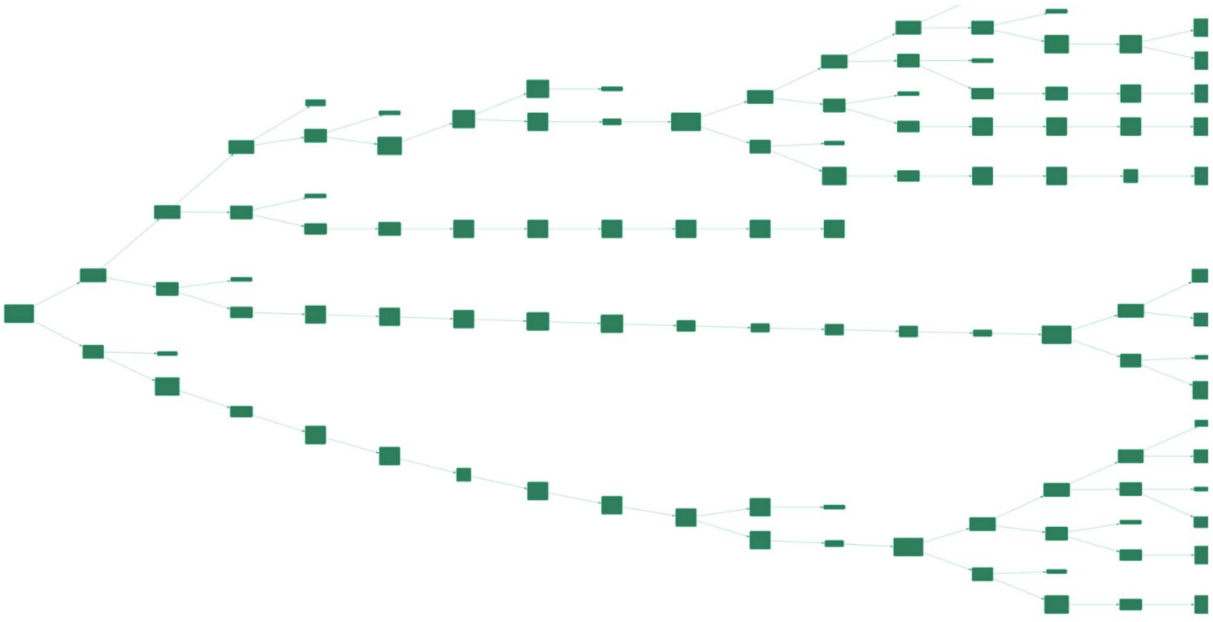
Instructions MVE	Description
STOP	Fin d'exécution normale sans retour de données. L'excédent d'essences est retourné à l'utilisateur.
RETURN	Fin d'exécution normale avec retour de données. L'excédent d'essences est retourné à l'utilisateur.
REVERT	Fin d'exécution anormale due à des conditions non respectées. L'excédent d'essences est retourné à l'utilisateur.
PUSH0 JUMP	Exception volontaire. La commande « throw » dans Solidity est compilée à bas niveau à un saut à une adresse inexistante, représentée par l'adresse 0. L'excédent d'essences n'est pas retourné à l'utilisateur.
ASSERT_FAIL	Fin d'exécution anormale due à des conditions non respectées. L'excédent d'essences n'est pas retourné à l'utilisateur.
SELFDESTRUCT	Destruction du contrat intelligent. Les données sont effacées et le contrat ne peut plus être appelé.

Chaque détection de ces instructions représente la fin d'un chemin d'exécution. De plus, le repérage de la combinaison « PUSH0 » et « JUMP » ainsi qu'« ASSERT_FAIL » classifient automatiquement le chemin comme étant frauduleux puisque ces cas terminent abruptement l'exécution et consomment toute l'essence de l'utilisateur. À noter que la commande « throw » de Solidity était compilée à « PUSH0 » et « JUMP », mais est maintenant compilée à « REVERT » depuis la version 0.4.10 de Solidity. « REVERT » retourne l'excédent d'essences et n'est pas considéré comme frauduleux.

4.2.2 Caractérisation des bifurcations frauduleuses

Un contrat intelligent, comme tout programme d'une certaine complexité, possède plusieurs chemins d'exécution. L'exécution symbolique permet de les détecter et de les représenter sous forme de graphe de contrôle tel qu'illustré dans la figure 4-1.

Figure 4-1: Exemple d'un graphe de contrôle généré par Mythril



Dans la figure 4-1, les nœuds représentent les états d'exécution et sont reliés par des arcs. Chaque arc est une condition à respecter définie par un prédicat. Un prédicat de chemin est alors l'addition logique de tous les prédicats sur un chemin d'exécution donné.

Pour déceler les bifurcations liées à un comportement de vol d'essences, les prédicats de chemin sont analysés. Si ceux-ci possèdent des bifurcations basées sur des instructions qui sont reliées au fonctionnement des chaînes de blocs, le chemin est alors classifié comme potentiellement frauduleux. L'analyse du coût en essences déterminera si ce chemin est réellement frauduleux. Le tableau 4-3 détaille les instructions de la MVE qui sont liées au fonctionnement des chaînes de blocs et que cet essai tentera de déceler dans les prédicats de chemin.

Tableau 4-3 : Instructions frauduleuses si détectées dans un prédicat de chemin

Instructions MVE	Description	Explication
COINBASE	Adresse du bénéficiaire du bloc	Le mineur qui résout en premier le casse-tête cryptographique devient le bénéficiaire du bloc et reçoit une récompense.
DIFFICULTY	Difficulté du bloc courant	La difficulté du casse-tête cryptographique varie constamment en fonction du nombre de mineurs connectés au réseau.
BLOCKHASH	Hachage d'un bloc	L'ordre d'ajout d'un bloc et son contenu est imprévisible. Cela dépend du bénéficiaire et des transactions qu'il a décidé d'ajouter.
NUMBER	Nombre du bloc courant	
TIMESTAMP	Date et heure d'ajout du bloc au registre de données	Un bloc est ajouté lorsque le casse-tête cryptographique est résolu et ne dépend pas d'un intervalle précis.

4.2.3 Prédiction du prix d'un chemin d'exécution en essences

L'exécution de chaque instruction par la MVE coûte une quantité prédéfinie d'essences [9]. Cependant, la prédiction de ce coût à l'avance n'est pas simple. En effet, un contrat intelligent peut posséder plusieurs chemins d'exécution qui dépendent de l'état des chaînes de blocs et des paramètres d'entrée. De plus, puisque la MVE est un système Turing-complet, les boucles et les fonctions récursives sont permises ce qui complexifie la tâche. Par exemple, le compilateur Solc contient un outil d'estimation d'essences, mais retourne « infini » lorsque la fonction analysée renferme une boucle. Albert [36] a tenté de résoudre ce problème et a créé l'outil Gastap pour déterminer la quantité maximale d'essences. Cependant, au moment de la rédaction de l'essai, Gastap n'est pas encore disponible pour utilisation. Quant à Oyente, il ne

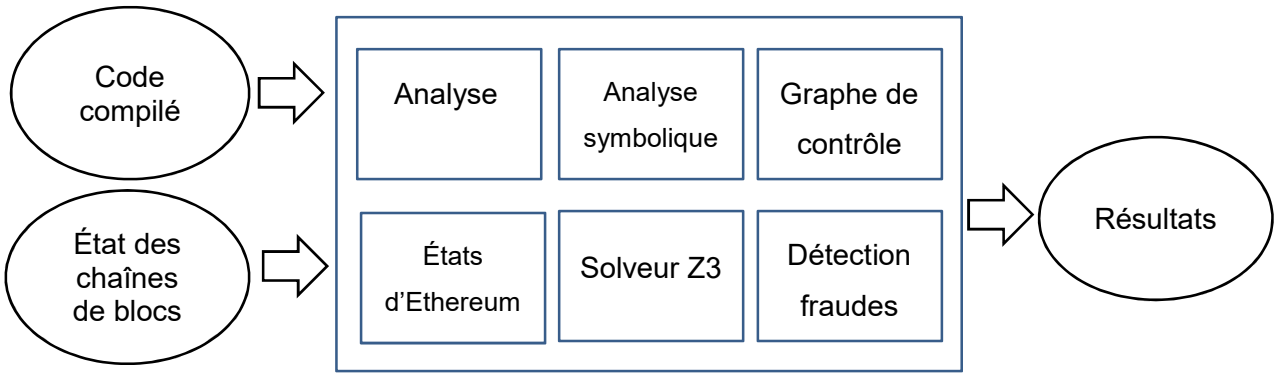
possède pas de fonctionnalité pour calculer l'essence et n'a pas été mis à jour depuis plus d'un an. Heureusement, Mythril a reçu une mise à jour en novembre 2018 lui permettant d'estimer la quantité d'essences utilisées pour chaque état d'exécution. L'outil effectue cependant un calcul simple et retourne la limite minimale et maximale d'essences. Pour cette raison, l'outil Mythril est retenu pour la réalisation de cet essai.

4.3 Création de l'instrument de mesure Sherlock

L'outil de vérification symbolique Mythril est apte à déceler plusieurs failles majeures dans les contrats intelligents [3]. Ce dernier a été conçu avec une approche modulaire pour encourager l'ajout de nouvelles fonctionnalités. En effet, Mythril est libre de droits et le code source est accessible sur GitHub [54]. Pour la deuxième étape de l'expérimentation, cet outil est modifié pour permettre la collecte des mesures nécessaires à la recherche. L'ajout d'un nouveau module représente l'approche retenue pour ajouter des fonctionnalités. Le nouvel instrument de mesure ainsi créé est alors nommé Sherlock.

La figure 4-2 expose les différentes composantes de Sherlock ainsi que les paramètres d'entrée et de sortie. Comparée à Mythril, la nouveauté réside dans l'ajout du module « Détection fraudes ». Aussi, Sherlock retourne comme résultats les mesures décrites dans le tableau 4-1 ainsi que la mention « frauduleux » ou « non frauduleux » pour chaque contrat intelligent analysé.

Figure 4-2 : Architecture de Sherlock



Le développement de Sherlock requiert plusieurs étapes :

1. Créer un environnement de développement :
    a. Installation d'Ubuntu WSL (Windows Subsystem for Linux).
    b. Installation de Truffle [55], un cadriciel pour le développement et le déploiement de contrats intelligents.
    c. Installation de Ganache [56], une chaîne de blocs Ethereum locale.
    d. Installation de Python 3.
    e. Installation de Visual Studio Code et d'un module pour la programmation en Solidity.
2. Télécharger le code source de Mythril de GitHub et installer les dépendances.
    a. Lien GitHub : https://github.com/ConsenSys/mythril-classic
3. Analyser le code source et le fonctionnement de Mythril.
4. Développer et intégrer le nouveau module « Détection fraudes ». Le langage de développement utilisé est Python 3.

Une fois Sherlock développé, la prochaine activité consiste à valider cet outil pour s'assurer qu'il soit en mesure de détecter les types de vol d'essences décrits dans cet essai.

4.4 Validation du fonctionnement de Sherlock

Avant d'utiliser le nouvel outil de mesure Sherlock sur des contrats intelligents d'Ethereum, le fonctionnement de celui-ci doit être validé. Pour réaliser cette tâche, les étapes suivantes sont effectuées :

1. Créer sept contrats intelligents qui possèdent les comportements de vol d'essences.
    a. Chaque cas de vol d'essences est représenté :
        i. Présence de la combinaison « PUSH0 » et « JUMP ».
        ii. Présence de l'instruction « ASSERT_FAIL ».
        iii. Présence d'une des cinq instructions frauduleuses, lorsque détectées dans un prédicat de chemin.
    b. Déployer les sept contrats intelligents sur une plateforme test d'Ethereum.
        i. La chaîne de blocs locale Ganache est utilisée.

    c. Invoquer les contrats intelligents avec les critères nécessaires pour déclencher les chemins considérés comme frauduleux.
2. Créer deux contrats intelligents qui n'ont pas de comportement de vols d'essences.
    a. Déployer les deux contrats intelligents sur la plateforme test.
    b. Invoquer les contrats intelligents à plusieurs reprises pour déterminer qu'il n'y a pas de chemins considérés comme frauduleux.
3. Compiler le code source des contrats intelligents en code intermédiaire.
    a. Depuis la version 0.4.10 du compilateur de Solidity, la combinaison « PUSH0 » et « JUMP » a été retirée en faveur de « REVERT ». Pour ce cas de vol d'essences, la version 0.4.9 du compilateur est utilisée. Pour les autres contrats intelligents, la version 0.5.20 du compilateur est en usage.
4. Utiliser Sherlock pour examiner les codes intermédiaires.
5. Cueillir et analyser les résultats.
6. Apporter les améliorations nécessaires au module « Détection fraudes » et recommencer les étapes 4 et 5 jusqu'à ce que les résultats soient satisfaisants. En effet, Sherlock doit être en mesure de détecter les contrats intelligents frauduleux créés et ne pas donner de faux résultats positifs pour les deux contrats intelligents non frauduleux.

Cette activité permet de vérifier que Sherlock catégorise adéquatement les contrats intelligents frauduleux et non frauduleux. Une fois la validation terminée, l'outil est désormais prêt pour examiner des contrats intelligents réels.

4.5 Échantillonnage des contrats intelligents

Une fois que Sherlock répond aux exigences, cet outil est utilisé sur des contrats intelligents existants. Toutefois, plus d'un million de contrats intelligents sont déployés dans l'environnement de production d'Ethereum [21]. Compte tenu de la nature des chaînes de blocs, les codes intermédiaires de toutes ces applications décentralisées sont accessibles.

Cet essai cible spécifiquement les contrats intelligents dont le code source est accessible. En effet, pour avoir la certitude que Sherlock retourne des résultats exacts, il est nécessaire d'effectuer une inspection manuelle du code source.

Etherscan.io rendait disponible le code source de plusieurs milliers de contrats intelligents. Cependant, ce site Internet a modifié son fonctionnement et n'affiche maintenant que le code source des 2000 contrats intelligents les plus récents [57]. Heureusement, lors de son analyse, Hegedűs [58] a pu recueillir le code source de plus de 40 000 contrats intelligents avant ce changement et les a publiés sur GitHub [59].

En raison de contraintes de temps, un échantillonnage de la population accessible est réalisé. Pour diminuer les risques de biais, l'approche de l'échantillonnage probabiliste est choisie, plus particulièrement la sélection aléatoire systématique. Dans la recherche de Hegedűs, les contrats intelligents sont ordonnés de 1 à 40 352. Un contrat intelligent est alors sélectionné par bloc de 1000 contrats, ce qui génère un ensemble de 40 éléments. Cette sélection est exécutée à un point précis dans le temps, avant le début de l'expérimentation.

L'analyse de 40 contrats intelligents est suffisante pour déterminer l'efficacité de l'exécution symbolique pour la détection de fraudes d'essences. Un échantillonnage plus élevé améliorerait la précision des résultats, mais serait également plus exigeant en temps. En effet, pour valider les résultats, une inspection manuelle du code source des contrats intelligents choisis doit être réalisée.

Selon Tonelli [60], les contrats intelligents possèdent entre 2 et 4241 lignes avec une moyenne de 316 lignes. Ceux qui détiennent moins de 10 lignes sont exclus de l'échantillonnage puisqu'ils ne possèdent pas assez de logique pour être pertinents. De plus, s'ils ont plus de 2000 lignes, ils sont également exclus en raison de contraintes de temps. Le prochain candidat dans la liste est alors sélectionné.

Le tableau 4-4 résume l'approche utilisée pour l'échantillonnage dans l'essai.

Tableau 4-4 : Approche pour l'échantillonnage

	Nombre de contrats intelligents	Justification
Population totale	> 1 000 000	Selon Etherscan.io [21]
Population cible	40 352	Contrats intelligents avec code source recueillis par Hegedűs [59]
Population accessible	40 352	

Population accessible après exclusion	Entre 32 281 et 40 352	Critères d'exclusion : < 10 lignes et > 2 000 lignes. Selon Tonelli [60], 80 % des contrats intelligents possèdent entre 52 et 720 lignes.
Échantillonnage par sélection aléatoire systématique	40	Un contrat intelligent est sélectionné par bloc de 1000 dans la liste.

Une fois que les éléments de l'échantillonnage sont choisis, ceux-ci sont examinés avec l'outil Sherlock et les résultats sont recueillis.

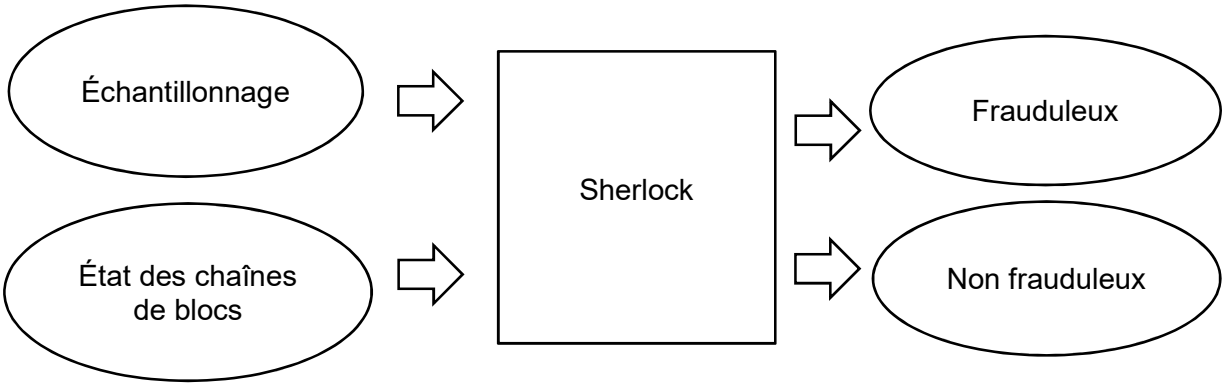
4.6 Collecte des données

Pour répondre à la question de recherche, deux types de données sont requis :

- Les résultats de l'analyse par l'exécution symbolique avec Sherlock;
- Les résultats de l'inspection manuelle du code source.

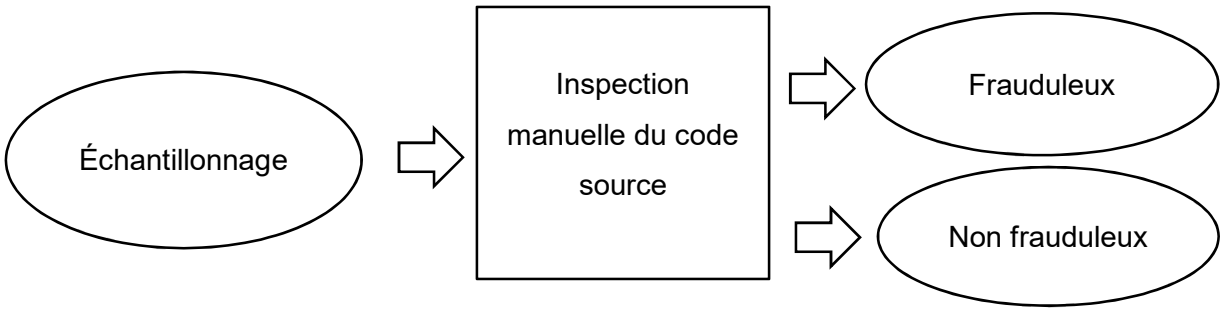
En premier lieu, le code intermédiaire des contrats intelligents dans l'échantillonnage est extrait d'Ethereum puis soumis comme paramètre d'entrée dans Sherlock. Les 40 contrats intelligents sont alors examinés et répertoriés dans une des deux catégories suivantes : frauduleux ou non frauduleux. La figure 4-3 schématise ce processus de collecte de données.

Figure 4-3 : Collecte des résultats par l'exécution symbolique



En deuxième lieu, le code source de chaque contrat intelligent de l'échantillonnage est inspecté manuellement et est également catégorisé comme étant frauduleux ou non. La figure 4-4 décrit cette méthode de collecte de données.

Figure 4-4 : Collecte des résultats par l'inspection manuelle



Une fois la collecte des données effectuée, il est probable qu'aucun contrat intelligent frauduleux n'ait été sélectionné. Pour pallier ce manque, l'approche suivante sera mise en œuvre :

- Sélection de 40 nouveaux contrats intelligents par sélection aléatoire systématique;
- Analyse par Sherlock de ces nouveaux cas. Conserver les contrats intelligents catégorisés comme étant frauduleux;
- Effectuer l'inspection manuelle du ou des contrats intelligents frauduleux selon Sherlock;
- S'il n'y a toujours pas de contrats intelligents frauduleux, des fraudes d'essences seront injectées dans des contrats intelligents non frauduleux. Ils seront par la suite examinés par Sherlock.

4.7 Analyse des résultats

La recherche prédit que l'efficacité de l'exécution symbolique est d'au moins 80 % pour la détection de fraudes d'essences dans les contrats intelligents. Pour valider cette hypothèse, les résultats recueillis précédemment sont analysés statistiquement.

La première activité consiste à statuer sur l'efficacité de l'exécution symbolique pour les bonnes détections en comparant les résultats de l'inspection manuelle et de l'outil Sherlock. Pour

chaque contrat intelligent frauduleux détecté par l'inspection manuelle, Sherlock doit également fournir le même résultat pour une efficacité de 100 %. Sinon, un ratio est effectué selon la formule suivante :

$$\%\ d'efficacité\ de\ bonnes\ détections -\ cas\ frauduleux\ = \frac{\#\ \text{bonnes détections contrats intelligents frauduleux (exécution symbolique)}}{\#\ \text{contrats intelligents frauduleux (inspection manuelle)}}$$

Similairement, le pourcentage d'efficacité pour la détection des cas non frauduleux est calculé selon la formule suivante :

$$\%\ d'efficacité\ de\ bonnes\ détections -\ cas\ \ non\ frauduleux = \frac{\#\ \text{bonnes détections contrats intelligents non frauduleux (exécution symbolique)}}{\#\ \text{contrats intelligents non frauduleux (inspection manuelle)}}$$

Pour avoir l'efficacité totale, il suffit d'additionner les bonnes détections et de faire le ratio avec la quantité de contrats intelligents dans l'échantillonnage :

$$\%\ d'efficacité = \frac{\#\ \text{bonnes détections contrats intelligents (exécution symbolique)}}{\#\ \text{contrats intelligents analysés}}$$

L'efficacité prévue de l'exécution symbolique n'est pas de 100 % puisque la méthode est susceptible de retourner des faux résultats. En effet, des contrats intelligents peuvent être catégorisés comme frauduleux par Sherlock, mais ne le sont pas en réalité. Luu a eu cette problématique et a chiffré le pourcentage de faux positifs d'Oyente à 6,4 % [1]. Dans cet essai, le taux de faux résultats est chiffré comme étant le pourcentage de faux positifs additionné au pourcentage de faux négatifs. Les formules suivantes sont alors utilisées :

$$\%\ faux\ positifs = \frac{\#\ \text{contrats intelligents frauduleux selon Sherlock,mais pas selon l}'\text{inspection manuelle}}{\#\ \text{contrats intelligents analysés}}$$

$$\%faux\ négatifs = \frac{\#\ \text{contrats intelligents non frauduleux selon Sherlock,mais frauduleux selon l}'\text{inspection manuelle}}{\#\ \text{contrats intelligents analysés}}$$

$$\%\ faux\ résultats = \%\ \text{faux positifs} + \%\ \text{faux négatifs}$$

4.8 Limites

La démarche expérimentale décrite dans ce chapitre n'est pas sans failles et comporte plusieurs limites. En effet, certains inconvénients sont exprimés ci-dessous quant à la taille de l'échantillonnage, la méthode de sélection et de validation.

Premièrement, une quantité plus élevée d'échantillons garantirait des résultats plus précis. Cependant, le temps nécessaire pour la validation par inspection manuelle empêche un échantillonnage plus nombreux.

Deuxièmement, l'inspection manuelle des contrats intelligents sélectionnés est essentielle pour s'assurer que ceux-ci possèdent les comportements frauduleux ciblés. En effet, Luu [1] s'est servi de cette technique pour établir le taux d'efficacité d'Oyente malgré l'effort requis. Une autre approche pour l'analyse des contrats intelligents a été considérée, mais n'a pas été retenue. En effet, plusieurs compagnies offrent un service d'audit pour les contrats intelligents. Toutefois, cette façon de procéder comporte plusieurs désavantages :

- Les failles majeures sont ciblées, mais il n'y a aucune mention des vols d'essences;
- La méthode d'analyse n'est pas toujours connue;
- Cette approche engendre un coût financier.

Troisièmement, l'échantillonnage est réalisé à un point précis dans le temps. Ceci peut être problématique puisque la quantité de contrats intelligents ne cessent d'augmenter. Par exemple, Luu n'a comptabilisé que 175 contrats avec code source en 2016 [1] alors qu'il y en a plus de 40 352 au moment de l'essai. L'échantillonnage effectué pourrait ne plus être représentatif entre le moment de sélection et le moment de publication de l'essai.

Finalement, rien ne garantit que les échantillons sélectionnés contiennent des contrats intelligents frauduleux. Dans une telle situation, de nouveaux éléments devront être choisis. Dans le cas où il n'y aurait toujours pas de fraudes d'essences, des contrats existants seront modifiés pour les rendre frauduleux.

4.9 Facteurs de succès

Pour que la recherche soit un succès, plusieurs facteurs rentrent en compte. Tout d'abord, les techniques de l'exécution symbolique doivent être bien comprises pour permettre le développement de Sherlock. En effet, Mythril comprend plusieurs modules dont le solveur Z3 et un constructeur de graphes qu'il faut maîtriser.

De plus, la recherche doit être accomplie dans un laps de temps court sous risque de perdre de la pertinence. En effet, le nombre de contrats intelligents augmente très rapidement. L'échantillonnage effectué avant le début de l'expérimentation risque de ne pas être représentatif de la situation une fois la recherche terminée.

4.10 Conclusion

Ce chapitre a décrit les différentes étapes à accomplir pour permettre de répondre à la question de recherche. Les principales démarches consistent à développer un nouvel outil de mesure nommé Sherlock, à effectuer l'échantillonnage des contrats intelligents par sélection aléatoire systématique, à effectuer la collecte des données et à valider manuellement les résultats. Les méthodes de l'expérimentation comprennent quelques inconvénients qui sont principalement liés à la taille de l'échantillonnage limitée. En effet, la technique de validation utilisée est l'inspection manuelle qui demande beaucoup de temps. Finalement, la recherche doit être réalisée dans un délai raisonnable pour rester pertinente dans un domaine qui évolue à grande vitesse.

Chapitre 5

Analyse des résultats

L'expérimentation dans cet essai consiste de plusieurs étapes. Tout d'abord, Sherlock a été construit sous la forme d'un module ajouté à Mythril. Ce nouvel outil est en mesure de détecter les instructions et les bifurcations frauduleuses dans l'environnement de développement sur des contrats intelligents fictifs. Le code source de Sherlock est partagé sur GitHub [4]. Les étapes subséquentes de l'expérimentation sont décrites dans ce chapitre, soit l'échantillonnage, la collecte des données et l'analyse des résultats. Finalement, une réponse à la question de recherche est fournie ainsi que les limites de la méthode.

5.1 Sélection de l'échantillonnage

Comme mentionné à la section 4.5, l'échantillonnage est réalisé à partir des 40 352 contrats intelligents recueillis par Hegedűs [58]. Ceux-ci sont publiés sur GitHub [59] et sont regroupés par lot de 10 000 dans quatre archives zip. Après avoir téléchargé et décompressé ces quatre fichiers, un contrat intelligent est alors sélectionné par bloc de 1000. Cependant, lors de la cueillette, des échantillons ont été exclus et le prochain candidat dans la liste est alors retenu. Le tableau 5-1 affiche les cas rejetés ainsi que la raison de l'exclusion.

Tableau 5-1 : Contrats intelligents rejetés lors de l'échantillonnage

Nom	Nombre de lignes	Raison de l'exclusion
lockEtherPay	109	Ce contrat intelligent est revenu à quatre reprises lors de l'échantillonnage. Le même code source a été déployé à différentes adresses dans Ethereum. Seule la première occurrence a été retenue.
lockEtherPay	109	
lockEtherPay	109	
EthTranchePricing	958	Ce contrat intelligent est revenu à deux reprises lors de l'échantillonnage. La première occurrence seulement a été retenue.
DataController	2413	Ce contrat intelligent a été exclu puisqu'il contient plus de 2000 lignes (critère d'exclusion défini à la section 4.5).

La liste complète de l'échantillonnage est disponible dans l'Annexe II.

5.2 Caractéristiques de l'échantillonnage

5.2.1 Catégorisation des échantillons

Bien que l'échantillonnage ne comporte que 40 éléments, celui-ci est tout de même varié. En effet, après inspection du code source, on y retrouve cinq types de contrats intelligents différents. Le tableau 5-2 affiche la répartition des échantillons recueillis par catégorie.

Tableau 5-2 : Répartition des échantillons par catégorie

Catégorie	Quantité d'échantillons	Définition de la catégorie
Monnaie	30	Contrat intelligent dont la vocation est la création ou la gestion d'une cryptomonnaie.
Librairie	4	Contrat intelligent contenant des classes et des fonctions destinées à être importées ou héritées.
Jeu	3	Contrat intelligent implémentant un jeu.
Protocole	1	Contrat intelligent implémentant un protocole de communication.
Loterie	2	Contrat intelligent implémentant une loterie.

La majorité des contrats intelligents dans l'échantillonnage implémentent une cryptomonnaie.

5.2.2 Quantité de lignes des échantillons

Le nombre de lignes du code source des échantillons varie grandement, allant d'un minimum de 32 lignes jusqu'à un maximum de 1605 lignes. La moyenne se situe à 293 lignes. En comparaison, Tonelli [60] a calculé dans sa recherche que les contrats intelligents possèdent en moyenne 316 lignes. Le tableau 5-3 détaille les statistiques concernant la quantité de lignes de code pour l'échantillonnage.

Tableau 5-3 : Statistiques concernant le nombre de lignes de code pour l'échantillonnage

	Minimum	Maximum	Moyenne	Médiane	Écart type
Lignes de code	32	1605	293	167	337

L'écart type élevé à 337 est un indicateur que l'échantillonnage est varié.

5.2.3 Version du compilateur Solidity

À partir du code intermédiaire déployé sur les chaînes de blocs, la version du compilateur Solidity qui a été utilisée ne peut pas être déduite. En effet, cette information n'est pas conservée. Cependant, l'analyse du code source permet de restreindre dans un intervalle la version du compilateur en usage. Cette donnée représente une indication de l'âge des programmes. Le tableau 5-4 synthétise les versions du compilateur Solidity supportées par les contrats intelligents de l'échantillonnage.

Tableau 5-4 : Compilateur Solidity supporté par l'échantillonnage

Versions du compilateur Solidity supportées	Nombre de contrats intelligents
>= 0.4.2 et < 0.5.0	1
>= 0.4.10 et < 0.5.0	5
>= 0.4.11 et < 0.5.0	4
>= 0.4.13 et < 0.5.0	1
>= 0.4.16 et < 0.5.0	4
>= 0.4.18 et < 0.5.0	12
>= 0.4.19 et < 0.5.0	2
>= 0.4.20 et < 0.5.0	1
>= 0.4.21 et < 0.5.0	1
>= 0.4.22 et < 0.5.0	1
>= 0.4.23 et < 0.5.0	1
>= 0.4.24 et < 0.5.0	2
>= 0.4.4 et < 0.5.0	1
>= 0.4.8 et < 0.5.0	1
< 0.5.0	1
0.4.18	1
0.4.21	1
0.4.24	1

Les données dans le tableau 5-4 sont déterminées principalement grâce aux annotations contenues dans le code source. Ainsi, la mention « pragma solidity ^0.4.18 » signifie que le programme doit être compilé au minimum avec la version 0.4.18, mais n'est pas compatible avec les versions 0.5.x et ultérieures. Toutefois, certains contrats intelligents ne contiennent pas d'annotations. La version du compilateur supportée peut quand même être déduite en

repérant certaines commandes dans le code source. Par exemple, la fonction « assert » a été introduite dans la version Solidity 0.4.10 [61]. De plus, aucun échantillon ne peut être compilé avec la version 0.5.x qui a été introduite en novembre 2018 [62]. Celle-ci apporte plusieurs changements majeurs, notamment la façon de déclarer le constructeur et les modificateurs d'accès pour les fonctions [63].

5.3 Résultats obtenus

Les échantillons obtenus dans la section précédente sont maintenant inspectés manuellement puis analysés par Sherlock.

5.3.1 Inspection manuelle du code source

Le code source de chacun des contrats intelligents est inspecté manuellement dans le but de déceler les instructions ainsi que les bifurcations frauduleuses. En tout, 11 736 lignes de code ont été vérifiées. Le tableau 5-5 présente la quantité d'échantillons frauduleux et non frauduleux détectés par l'inspection manuelle et catégorisés par type de contrat intelligent.

Tableau 5-5 : Échantillons frauduleux et non frauduleux détectés par l'inspection manuelle

Type	Frauduleux	Non Frauduleux
Monnaie	15	15
Jeu	3	0
Librairie	3	1
Protocole	0	1
Loterie	0	2
Total	**21**	**19**

L'inspection manuelle a repéré un total élevé de cas frauduleux, soit 21 pour un ratio de 52,5 % des échantillons. Les types de fraudes détectées sont affichés dans le tableau 5-6.

Tableau 5-6 : Types de fraudes détectées par l'inspection manuelle

Type Fraude	Quantité de fraudes
Instruction « ASSERT_FAIL »	21
Bifurcation « COINBASE »	0
Bifurcation « DIFFICULTY »	0
Bifurcation « BLOCKHASH »	0
Bifurcation « NUMBER »	0
Bifurcation « TIMESTAMP »	1
Total	**22**

Le nombre de fraudes décelées semble élevé à première vue. Celles-ci sont principalement causées par une fin anormale de l'exécution avec l'instruction « ASSERT_FAIL ». En effet, l'inspection a révélé deux cas de division par zéro, deux indices de tableau invalides, une mauvaise déclaration d'objet et 15 assertions non respectées. De plus, un échantillon a été compilé avec une ancienne version de Solidity (annotation ^ 0.4.2) qui génère des exceptions lors d'erreurs dans les appels de fonction. Plus de détails concernant l'instruction « ASSERT_FAIL » sont donnés à la section 5.5.1. Finalement, seule une bifurcation frauduleuse a été détectée soit la bifurcation « TIMESTAMP ». Aussi, le contrat intelligent « EthTranchePricing » possède deux types de fraudes ce qui explique le total de 22. L'Annexe III détaille les résultats de l'inspection manuelle.

5.3.2 Analyse par Sherlock - estimation du coût en essences

Pour donner suite à l'inspection manuelle qui a trouvé plusieurs fraudes, le code intermédiaire des échantillons est extrait d'Ethereum puis analysé par Sherlock. Ce dernier tire profit de la fonction d'estimation d'essences de Mythril pour calculer le coût d'un chemin d'exécution. Cependant, lors de l'expérimentation, les estimations d'essences fournies par l'outil ne sont pas assez précises pour être utile. En effet, de grandes variations existent entre le coût minimal et le coût maximal d'essences pour un chemin donné. Lors de la phase de test, ces variations étaient de moindre envergure puisque les contrats intelligents créés étaient de nature moins complexe. Le tableau 5-7 affiche les estimations d'essences pour la fonction « xorReduce » dans le contrat « Bytes32Lib ».

Tableau 5-7 : Estimations d'essences pour la fonction xorReduce dans Bytes32Lib

Instruction de fin	Coût minimal en essences	Coût maximal en essences	Vol d'essences
ASSERT_FAIL	302	3077	Instruction « ASSERT_FAIL »
RETURN	432	3581	Aucun
ASSERT_FAIL	408	3369	Instruction « ASSERT_FAIL »
ASSERT_FAIL	809	4805	Instruction « ASSERT_FAIL »
ASSERT_FAIL	813	4809	Instruction « ASSERT_FAIL »

Pour le chemin d'exécution non frauduleux se terminant avec l'instruction « RETURN », le coût en essences varie de 432 à 3581, soit un facteur de huit entre le minimum et le maximum. Il est alors difficile de comparer les coûts d'exécution d'un chemin à un autre. Dans la prochaine section, pour déterminer si un chemin est frauduleux ou non, la cause de la bifurcation ainsi que la présence de l'instruction « ASSERT_FAIL » seront les critères adoptés.

5.3.3 Résultats de l'analyse symbolique par Sherlock

Le langage Solidity permet l'usage de boucles et de méthodes récursives ce qui amène la possibilité d'une quantité exponentielle de chemins d'exécution. Pour conserver le temps d'analyse raisonnable, la profondeur de récursion de l'exécution symbolique est configurable dans Sherlock. Dans cet essai, trois profondeurs de récursion différentes sont utilisées : 20, 50 et 100. Par défaut, Mythril établit ce paramètre à 22 [64]. Après l'analyse, Sherlock retourne comme résultat si l'échantillon est frauduleux ou non ainsi que la fraude détectée. Le tableau 5-8 présente les résultats de Sherlock en fonction de la profondeur de récursion (PR).

Tableau 5-8 : Échantillons frauduleux et non frauduleux détectés par Sherlock par profondeur de récursion (PR)

Type	Frauduleux (PR=20)	Non Frauduleux (PR=20)	Frauduleux (PR=50)	Non Frauduleux (PR=50)	Frauduleux (PR=100)	Non Frauduleux (PR=100)
Monnaie	16	14	16	14	16	14
Jeu	1	2	2	1	3	0
Librairie	3	1	3	1	3	1
Protocole	1	0	1	0	1	0
Loterie	1	1	1	1	1	1
Total	**22**	**18**	**23**	**17**	**24**	**16**

Sherlock a décelé 24 contrats intelligents frauduleux avec une profondeur de récursion de 100, soit 60 % de l'échantillonnage. Avec une profondeur de récursion de 20, l'outil retourne 55 % de cas frauduleux. Ces résultats sont plus élevés que l'inspection manuelle qui a détecté 21 cas frauduleux. L'outil est également en mesure d'identifier le type de fraude, tel que détaillé dans le tableau 5-9.

Tableau 5-9 : Types de fraudes détectées par Sherlock par profondeur de récursion (PR)

Type Fraude	Quantité de fraudes (PR=20)	Quantité de fraudes (PR=50)	Quantité de fraudes (PR=100)
Instruction « ASSERT_FAIL »	17	18	19
Bifurcation « COINBASE »	0	0	0
Bifurcation « DIFFICULTY »	0	0	0
Bifurcation « BLOCKHASH »	0	0	0
Bifurcation « NUMBER »	1	1	1
Bifurcation « TIMESTAMP »	5	5	5
Total	**23**	**24**	**25**

Peu importe la profondeur de récursion, Sherlock est incapable de découvrir les 21 fraudes avec l'instruction « ASSERT_FAIL » trouvées par l'inspection manuelle. Il en repère 19 dans le meilleur des cas. De plus, l'outil fournit plusieurs faux résultats. En effet, ce dernier a détecté une bifurcation causée par l'instruction « NUMBER » et cinq bifurcations causées par l'instruction « TIMESTAMP », alors que l'inspection manuelle reporte seulement une bifurcation causée par « TIMESTAMP ». La méthode d'analyse n'est donc pas parfaite. Les résultats détaillés par échantillon de Sherlock sont présentés dans l'Annexe III. La prochaine section calcule le taux d'efficacité de l'outil.

5.4 Efficacité de Sherlock

Pour déterminer l'efficacité de Sherlock dans la détection des fraudes d'essences, les formules décrites à la section 4.7 sont mises en usage. En premier lieu, le pourcentage d'efficacité pour les bonnes détections des contrats frauduleux est calculé et décrit dans le tableau 5-10.

Tableau 5-10 : Pourcentage d'efficacité pour les bonnes détections de contrats frauduleux

Contrats frauduleux (inspection manuelle)	Bonnes détections par Sherlock pour les contrats frauduleux			Efficacité		
	PR = 20	PR = 50	PR = 100	PR = 20	PR = 50	PR = 100
21	17	18	19	81,0 %	85,7 %	90,5 %

En deuxième lieu, le pourcentage d'efficacité de Sherlock pour les bonnes détections de contrats non frauduleux est présenté dans le tableau 5-11.

Tableau 5-11 : Pourcentage d'efficacité pour les bonnes détections de contrats non frauduleux

Contrats non frauduleux (inspection manuelle)	Bonnes détections par Sherlock pour les contrats non frauduleux			Efficacité		
	PR = 20	PR = 50	PR = 100	PR = 20	PR = 50	PR = 100
19	14	14	14	73,7 %	73,7 %	73,7 %

En combinant les résultats précédents, l'efficacité totale de Sherlock est affichée dans le tableau 5-12.

Tableau 5-12 : Pourcentage d'efficacité totale de Sherlock

Contrats dans l'échantillonnage	Bonnes détections (Sherlock)			Efficacité		
	PR = 20	PR = 50	PR = 100	PR = 20	PR = 50	PR = 100
40	31	32	33	**77,5 %**	**80,0 %**	**82,5 %**

En fonction de la profondeur de récursion, l'efficacité totale de l'outil varie entre 77,5 % et 82,5 %. Sherlock n'est alors pas parfait. En effet, celui-ci retourne des résultats faux positifs et faux négatifs. Le tableau 5-13 expose le pourcentage de faux positifs alors que le tableau 5-14 montre le pourcentage de faux négatifs.

Tableau 5-13 : Pourcentage de résultats faux positifs retournés par Sherlock

Contrats dans l'échantillonnage	Résultats faux positifs par Sherlock			Pourcentage de résultats faux positifs		
	PR = 20	PR = 50	PR = 100	PR = 20	PR = 50	PR = 100
40	5	5	5	12,5 %	12,5 %	12,5 %

Tableau 5-14 : Pourcentage de résultats faux négatifs retournés par Sherlock

Contrats dans l'échantillonnage	Résultats faux négatifs par Sherlock			Pourcentage de résultats faux négatifs		
	PR = 20	PR = 50	PR = 100	PR = 20	PR = 50	PR = 100
40	4	3	2	10,0 %	7,5 %	5,0 %

Le pourcentage de résultats faux positifs de Sherlock est de 12,5 %, ce qui plus élevé que celui d'Oyente chiffré à 6,4 % dans la recherche de Luu [1]. De plus, Sherlock a un pourcentage de résultats faux négatifs entre 5,0 % et 10,0 % en fonction de la profondeur de récursion.

5.5 Analyse des résultats

L'inspection manuelle a révélé un haut taux de contrats intelligents frauduleux et l'outil Sherlock a pu déceler la majorité de ces fraudes. Cependant, certains cas n'ont pas pu être détectés et l'outil donne également des faux positifs. Cette section discute des fraudes d'essences décelées ainsi que les bonnes et mauvaises détections.

5.5.1 Taux élevé de fraudes avec l'instruction « ASSERT_FAIL »

L'inspection manuelle a décelé 21 fraudes avec l'instruction « ASSERT_FAIL ». De ce nombre, Sherlock en a repéré entre 17 et 19 en fonction de la profondeur de récursion. À première vue, la quantité de fraudes semble élevée. Cependant, elle est causée principalement par des erreurs de programmation. En effet, l'instruction « ASSERT_FAIL » est déclenchée dans les situations suivantes :

- Arrêt exceptionnel de l'exécution :
    - Division par zéro;

    - Indice de tableau hors portée;
    - Erreur dans la déclaration d'objet (pas de commande « new »);
- Assertion non respectée;
- Contrat intelligent compilé avec une version de Solidity antérieure à 0.4.10. Les erreurs lors de l'appel d'une fonction (par exemple un appel avec un mauvais argument) sont représentées par un chemin d'exécution qui se termine par « ASSERT_FAIL ». À partir de la version 0.4.10, ces erreurs sont compilées à « REVERT » qui arrête l'exécution, mais retourne l'essence restant.

Les arrêts exceptionnels et les assertions non respectées sont des cas qui ne devraient pas arriver lors de l'exécution. Le tableau 5-15 recèle les causes de l'instruction « ASSERT_FAIL » dans l'échantillonnage à la suite de l'inspection manuelle.

Tableau 5-15 : Causes de la présence de l'instruction « ASSERT_FAIL »

	Assertion non respectée	Indice de tableau hors portée	Erreur dans la déclaration d'objet	Version Solidity < 0.4.10	Division par zéro
Quantité de contrats intelligents	15	2	1	1	2

La cause principale de l'instruction « ASSERT_FAIL » est due à une mauvaise utilisation des assertions avec la fonction « assert ». En effet, celle-ci devrait être mise en usage lors de tests et pour valider des prédicats qui devraient toujours être vrais. Dans les contrats intelligents inspectés, les assertions sont utilisées pour arrêter l'exécution lorsqu'un dépassement d'entier survient. Effectivement, plusieurs programmes dans l'échantillonnage se servent d'une ancienne version de la librairie « SafeMath » développée par OpenZeppelin [65] pour traiter les dépassements d'entier. Voici un extrait de cette librairie du contrat intelligent « NePay » :

Code source de la libraire « SafeMath » dans « NePay »

```
library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a * b ;
    assert(a == 0 || c / a == b);
    return c;
```

```
  }

  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a / b ;
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b ;
  }

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c ;
  }
}
```

Par exemple, pour la fonction « sub », si le deuxième argument est plus grand que le premier, l'assertion devient fausse et l'exécution du contrat intelligent s'arrête abruptement en consommant toute l'essence de l'utilisateur. Une bonne implémentation de cette librairie serait de remplacer la fonction « assert » par la fonction « require ». Cette dernière termine également l'exécution, mais retourne le restant de l'essence à l'utilisateur. La version la plus récente de « SafeMath » a remplacé « assert » avec « require » et est accessible par GitHub [66]. Cette fraude n'est donc pas intentionnelle et relève plus d'une erreur de programmation. Les fonctions « assert » et « require » ont été introduites depuis la version Solidity 0.4.10 et ne sont pas toujours utilisées adéquatement. Cependant, que la fraude soit intentionnelle ou non, un appel à ces contrats intelligents dans Ethereum peut causer un vol d'essences.

Aussi, certains contrats intelligents possèdent une mauvaise implémentation de la librairie « SafeMath », mais ne sont pas considérés comme étant frauduleux. En effet, ceux-ci

possèdent des validations supplémentaires sur les données en entrée qui empêchent les dépassements d'entier. Les assertions dans « SafeMath » seront donc toujours vraies.

5.5.2 Résultats faux négatifs

L'outil Sherlock est incapable de détecter deux cas d'instruction « ASSERT_FAIL » avec une profondeur de récursion de 100, soit les contrats intelligents « Gdprcrowdsale » et « Yeedtoken ». Puisque ceux-ci ne contiennent pas ou peu de boucles, ils ont été analysés à nouveau par Sherlock avec une profondeur de récursion illimitée. Malgré ce changement, le résultat est resté le même. Ce sont donc deux cas qui ont été classés dans les faux négatifs et qui démontrent la limite de l'outil. D'autres résultats faux négatifs ont été relevés avec les profondeurs de récursion 20 et 50 et sont affichés dans le tableau 5-16.

Tableau 5-16 : Résultats faux négatifs par profondeur de récursion

Profondeur de récursion	Résultats faux négatifs détectés par Sherlock
20	Gdprcrowdsale, Yeedtoken, EtherScrolls, EtherGames
50	Gdprcrowdsale, Yeedtoken, EtherScrolls
100	Gdprcrowdsale, Yeedtoken

5.5.3 Résultats faux positifs

Sherlock a retourné cinq résultats faux positifs, soit quatre avec la bifurcation causée par l'instruction « TIMESTAMP » et un avec la bifurcation causée par l'instruction « NUMBER ». Le tableau 5-17 détaille les résultats faux positifs ainsi que la raison.

Tableau 5-17 : Résultats faux positifs et explications

Contrat intelligent	Résultats faux positifs détectés par Sherlock	Explications résultats faux positifs
NeobitTokens	Bifurcation « TIMESTAMP »	Donne un bonus de cryptomonnaies en fonction de la date d'achat.
LoopringProtocolImpl	Bifurcation « TIMESTAMP »	Validation de la date encapsulée dans la fonction « require ».

PixoArenaFounderToken	Bifurcation « TIMESTAMP »	Donne un bonus de cryptomonnaies en fonction de la date d'achat.
Lottery	Bifurcation « NUMBER »	L'instruction « Number » dans la bifurcation est utilisée à des fins de loterie.
MEMESCrowdsale	Bifurcation « TIMESTAMP »	Validation de la date encapsulée dans la fonction « require ».

Certaines cryptomonnaies accordent un bonus en fonction de la date d'achat. Cette situation crée un chemin d'exécution supplémentaire. Cependant, le coût en essences de cette bifurcation ne devrait pas varier énormément en comparaison avec les chemins avoisinants. Si la fonction d'estimation d'essences de Sherlock était plus précise, ce cas n'aurait pas été reporté comme frauduleux.

Les contrats « LoopringProtocolImpl » et « MEMESCrowdale » possèdent des validations impliquant la date de l'appel au programme. Cependant, celles-ci sont encapsulées avec « require » en début de fonction. Si ces validations ne sont pas respectées, l'exécution s'arrête immédiatement. De plus, l'essence de l'utilisateur est retournée, créant une perte minime. Encore une fois, ce cas n'aurait pas été reporté comme étant frauduleux si l'estimation d'essences de Sherlock était plus précise.

Finalement, le contrat « Lottery » est, comme son nom l'indique, une loterie. Ce dernier se sert de l'instruction « Number » pour déterminer un gagnant. Ce n'est donc pas une fraude d'essences.

5.5.4 Profondeur de récursion et temps d'exécution

Comme décrit à la section 5.4, l'efficacité de Sherlock varie en fonction de la profondeur de récursion. Cependant, le temps d'analyse s'accroît grandement lorsque ce paramètre est augmenté. Le tableau 5-18 affiche les statistiques concernant le temps d'exécution de Sherlock par rapport à la profondeur de récursion. Toute l'expérimentation a été effectuée sur un portable Microsoft Surface Laptop 2 qui possède un processeur quadricœur i5-8250u avec 8 Go de mémoire vive.

Tableau 5-18 : Temps d'exécution de Sherlock en fonction de la profondeur de récursion (PR)

	Moyenne	Médiane	Écart type	Minimum	Maximum
PR = 20	3 min 20 s	50 s	8 min 37 s	13 s	51 min 37 s
PR = 50	28 min 40 s	2 min 42 s	56 min 46 s	17 s	4 h 50 min 58 s
PR = 100	2 h 45 min 33 s	2 min 59 s	5 h 36 min 4 s	14 s	22 h 23 min 30 s

Lorsque la profondeur de récursion se situe à 20, le temps moyen d'exécution demeure raisonnable à 3 minutes et 20 secondes. Cependant, lorsque le paramètre est incrémenté, le temps d'exécution moyen augmente également très rapidement. Avec ce paramètre à 100, Sherlock prend en moyenne 2 heures et 45 minutes par échantillon. Après vérification, le temps d'exécution est peu affecté par la profondeur de récursion pour environ la moitié des échantillons, puisque ceux-ci ne contiennent pas ou peu de boucles et de fonctions récursives. En effet, la médiane est similaire lorsque la profondeur de récursion est saisie à 50 et à 100. Pour les contrats intelligents qui possèdent des boucles et des fonctions récursives, le temps d'exécution augmente exponentiellement, allant même jusqu'à presque une journée entière (22 heures et 23 minutes). L'écart type élevé illustre bien cette réalité. Il est alors nécessaire de bien définir ce paramètre en fonction du contrat intelligent à analyser.

5.6 Retour sur les hypothèses

L'hypothèse de départ stipulait que l'exécution symbolique pouvait détecter les fraudes de vols d'essences avec une efficacité d'au moins 80 %. Cette hypothèse est confirmée avec les résultats de Sherlock, à condition de paramétrer la profondeur de récursion adéquatement. En effet, il y a un compromis entre la profondeur de récursion et le temps d'exécution. Avec ce paramètre à 50, le taux d'efficacité de l'exécution symbolique se situe à 80 % avec un temps d'analyse moyen de 28 minutes et 40 secondes. L'efficacité augmente à 82,5 % si la profondeur de récursion est incrémentée à 100. Cependant, avec cette configuration, le temps d'analyse est très élevé.

Le taux d'efficacité pourrait être amélioré si Sherlock possédait un meilleur outil pour estimer le coût d'exécution en essences. Plusieurs résultats faux positifs auraient pu être évités. En dépit de tout cela, l'efficacité ainsi que le potentiel de l'exécution symbolique ont été démontrés dans le contexte de la détection des vols d'essences dans les contrats intelligents d'Ethereum.

5.7 Limite des résultats

Malgré les résultats positifs sur l'efficacité de la méthode, plusieurs limites sont présentes. Premièrement, tous les échantillons recueillis ont été développés et compilés avec une ancienne version du langage Solidity, c'est-à-dire la version 0.4.x. En effet, le bassin de contrats intelligents utilisé pour l'échantillonnage a été prélevé en août 2018. Depuis, une révision importante de Solidity a été rendue disponible le 13 novembre 2018 [62] avec la version 0.5.0. L'analyse n'a donc pas été réalisée sur les contrats intelligents les plus récents, ce qui constituait un des enjeux de l'essai.

Deuxièmement, l'expérimentation avec Sherlock a été effectuée avec seulement trois profondeurs de récursion, en raison de contraintes de temps. Il aurait été intéressant de voir les résultats de l'outil avec plus de points d'analyse.

Troisièmement, le temps de traitement pour un échantillon par Sherlock avec une profondeur de récursion de 100 peut prendre jusqu'à 22 heures dans le pire des cas. Dans cet essai, seulement 40 contrats intelligents ont été analysés. Cependant, si la quantité d'échantillons était plus élevée, ce paramètre n'aurait pas pu être utilisé. En effet, attribuable aux boucles et aux fonctions récursives, une explosion de chemin peut survenir et le temps d'analyse augmente alors exponentiellement. Ceci est une des limites de l'exécution symbolique.

Finalement, l'essai ne tient pas compte des appels que les contrats intelligents peuvent faire. Sherlock est en mesure de détecter si un appel est effectué, mais n'analyse pas le contrat appelé.

Conclusion

Le vol d'essences dans Ethereum est un phénomène peu discuté dans la littérature. Cependant, celui-ci cause de grands désagréments et mine la confiance des utilisateurs envers la technologie des chaînes de blocs. Pour remédier à ce problème, la technique de l'exécution symbolique a été mise à profit. L'objectif de la recherche était de valider si l'exécution symbolique pouvait déceler avec une efficacité d'au moins 80 % les contrats intelligents frauduleux.

Pour répondre à la question de recherche, la première étape consistait à caractériser un contrat intelligent frauduleux. Les mesures qui ont été utilisées sont le coût en essences d'un chemin d'exécution, la cause de la bifurcation ainsi que l'instruction causant la fin de l'exécution. Par la suite, un échantillonnage de 40 contrats intelligents d'Ethereum a été analysé par Sherlock. Cet outil exploite l'exécution symbolique et a été conçu pour détecter les comportements frauduleux caractérisés précédemment. Finalement, le code source des échantillons a été inspecté manuellement puis comparé aux résultats de Sherlock pour déterminer l'efficacité de l'outil.

Avec les résultats obtenus, l'hypothèse de départ a été validée puisque Sherlock possède la capacité de détecter les contrats intelligents frauduleux avec une efficacité maximale de 82,5 %. Cependant, des difficultés ont été rencontrées lors de l'expérimentation et l'exécution symbolique comporte des limites.

L'efficacité de la méthode dépend de la profondeur de récursion choisie qui affecte grandement la durée de l'analyse. Pour atteindre l'efficacité maximale de 82,5 %, une profondeur de récursion de 100 a été utilisée et le temps d'analyse moyen par cas s'élève à 2 heures et 45 minutes. Avec une profondeur de récursion à 50, l'efficacité est alors de 80 % avec un temps d'analyse moyen plus raisonnable de 28 minutes et 40 secondes. L'exécution symbolique requiert un temps d'analyse très élevé lorsque le programme possède des boucles et des méthodes récursives.

Également, la fonction d'estimation du coût en essences de Sherlock n'a pu être utilisée puisque celle-ci manquait de précision. La méthode de calcul étant trop simpliste, des variations considérables existaient entre les valeurs minimales et maximales d'essences retournées. Cette mesure a alors été ignorée et a causé une grande quantité de résultats faux positifs. En effet, Sherlock a fourni 12,5 % de résultats faux positifs comparés à 6,4 % pour Oyente [1].

De plus, Sherlock a été incapable de déceler deux fraudes, et ce peu importe la profondeur de récursion. Bien que l'exécution symbolique traverse et découvre tous les chemins d'exécution d'un programme, certains cas ont échappé à la méthode. Ces résultats démontrent une des limites de la méthode.

Plusieurs améliorations peuvent être apportées à l'expérimentation en commençant par l'analyse de contrats intelligents plus récents. En effet, au moment de la publication de l'essai, les échantillons auront été recueillis un an auparavant. Aucun n'a été développé avec la version 0.5.x de Solidity qui amène plusieurs changements majeurs. Puisque plusieurs fraudes sont causées par des erreurs de programmation, il serait intéressant de voir si les contrats intelligents plus récents répètent ces erreurs.

Une autre amélioration serait de tenir compte des appels effectués. En effet, un contrat intelligent peut appeler d'autres librairies ou des fonctions d'autres programmes. Le taux de fraudes détecté pourrait alors être plus élevé.

De plus, pour tenter d'améliorer l'efficacité des détections, quelques voies s'offrent à nous. La première serait d'améliorer l'outil d'estimation d'essences de Sherlock, ce qui réduirait la quantité de résultats faux positifs. Deuxièmement, Sherlock possède plusieurs paramètres qui peuvent être ajustés en plus de la profondeur de récursion. En effet, l'outil offre la possibilité d'analyser plusieurs exécutions consécutives du même contrat intelligent. Dans l'essai, chaque analyse représentait une exécution. Aussi l'algorithme d'usage pour la découverte de nœuds dans le graphe de contrôle peut être modifié. Dans l'expérimentation, l'algorithme par défaut a été utilisé, soit le parcours en profondeur. D'autres choix sont offerts tels le parcours en largeur et le parcours aléatoire.

Liste des références

[1] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, et A. Hobor, « Making Smart Contracts Smarter », dans *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, p. 254-269.

[2] T. Chen, X. Li, X. Luo, et X. Zhang, « Under-optimized smart contracts devour your money », dans *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, p. 442-446.

[3] « Mythril Classic: Security analysis tool for Ethereum smart contracts: ConsenSys/mythril-classic », 04-mai-2019. [En ligne]. Disponible à: https://github.com/ConsenSys/mythril-classic. [Consulté le: 04-mai-2019].

[4] H. Huynh, « Sherlock », *Sherlock Analyse Symbolique*, 14-oct-2019. [En ligne]. Disponible à: https://github.com/ppopops/Sherlock. [Consulté le: 16-oct-2019].

[5] J. Yli-Huumo, D. Ko, S. Choi, S. Park, et K. Smolander, « Where is current research on blockchain technology?—a systematic review », *PloS One*, vol. 11, nº 10, p. e0163477, 2016.

[6] S. Nakamoto, « Bitcoin: A Peer-to-Peer Electronic Cash System », 2008.

[7] « Bitcoin Hits a New Record High, But Stops Short of $20,000 », *Fortune*. [En ligne]. Disponible à: http://fortune.com/2017/12/17/bitcoin-record-high-short-of-20000/.

[8] « All Cryptocurrencies | CoinMarketCap ». [En ligne]. Disponible à: https://coinmarketcap.com/all/views/all/.

[9] G. Wood, « Ethereum: A secure decentralised generalised transaction ledger », *Ethereum Proj. Yellow Pap.*, vol. 151, 2014.

[10] D. Magazzeni, P. McBurney, et W. Nash, « Validation and Verification of Smart Contracts: A Research Agenda », *Computer*, vol. 50, nº 9, p. 50–57, 2017.

[11] « Someone Just Stole $50 Million from the Biggest Crowdfunded Project Ever. (Humans Can't Be Trusted) », *WIRED*. [En ligne]. Disponible à: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/. [Consulté le: 01-févr-2018].

[12] D. Chaum, « Blind Signatures for Untraceable Payments », dans *Advances in Cryptology*, Springer, Boston, MA, 1983, p. 199-203.

[13] W. Dai, « B-Money », 1998. [En ligne]. Disponible à: http://www.weidai.com/bmoney.txt.

[14] X. Li, P. Jiang, T. Chen, X. Luo, et Q. Wen, « A Survey on the security of blockchain systems », *Future Gener. Comput. Syst.*, août 2017.

[15] N. Szabo, « Formalizing and Securing Relationships on Public Networks », *First Monday*, vol. 2, nº 9, sept. 1997.

[16] « Solidity — Solidity 0.4.20 documentation ». [En ligne]. Disponible à: https://solidity.readthedocs.io/en/develop/. [Consulté le: 04-févr-2018].

[17] « One of Ethereum's Earliest Smart Contract Languages Is Headed for Retirement », *CoinDesk*, 02-août-2017. [En ligne]. Disponible à: https://www.coindesk.com/one-of-ethereums-earliest-smart-contract-languages-is-headed-for-retirement/. [Consulté le: 04-févr-2018].

[18] V. Buterin, « A next-generation smart contract and decentralized application platform », *White Pap.*, 2014.

[19] R. O'Connor, « Simplicity: A New Language for Blockchains », *ArXiv171103028 Cs*, p. 107-120, 2017.

[20] K. Delmolino, M. Arnett, A. Kosba, A. Miller, et E. Shi, « Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab », dans *International Conference on Financial Cryptography and Data Security*, 2016, p. 79–94.

[21] « Ethereum Accounts And Contracts ». [En ligne]. Disponible à: https://etherscan.io/accounts/c. [Consulté le: 29-mars-2018].

[22] « CRITICAL UPDATE Re: DAO Vulnerability », *Ethereum Blog*, 17-juin-2016. [En ligne]. Disponible à: https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/. [Consulté le: 01-févr-2018].

[23] Y. Hirai, « Defining the ethereum virtual machine for interactive theorem provers », dans *International Conference on Financial Cryptography and Data Security*, 2017, p. 520–535.

[24] D. P. Mulligan, S. Owens, K. E. Gray, T. Ridge, et P. Sewell, « Lem: reusable engineering of real-world semantics », ACM Press, 2014, p. 175-188.

[25] E. Hildenbrandt *et al.*, « KEVM: A Complete Semantics of the Ethereum Virtual Machine », 2017.

[26] « K Framework ». [En ligne]. Disponible à: http://www.kframework.org/index.php/Main_Page. [Consulté le: 25-févr-2018].

[27] K. Bhargavan, A. Delignat-Lavaud, et C. Fournet, « Formal Verification of Smart Contracts - Short Paper - F* », dans *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, p. 91-96.

[28] B. Mueller, « Smashing Ethereum Smart Contracts for Fun and Real Profit », présenté à 9th Annual HITB Security Conference, 2018, p. 54.

[29] A. Mense et M. Flatscher, « Security Vulnerabilities in Ethereum Smart Contracts », dans *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services - iiWAS2018*, Yogyakarta, Indonesia, 2018, p. 375-380.

[30] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, et M. Vechev, « Securify: Practical Security Analysis of Smart Contracts », dans *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, Toronto, Canada, 2018, p. 67-82.

[31] « New Technologies for the Blockchain: IELE (virtual machine) and K (universal language framework) | RV Blog ». [En ligne]. Disponible à: https://runtimeverification.com/blog/?p=459. [Consulté le: 05-févr-2018].

[32] D. Park, Y. Zhang, M. Saxena, P. Daian, et G. Roşu, « A formal verification tool for Ethereum VM bytecode », dans *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, Lake Buena Vista, FL, USA, 2018, p. 912-915.

[33] Melon, « Oyente Beta Release — KentRidge v0.1.0 », *Melonport Blog*, 19-juin-2017. [En ligne]. Disponible à: https://medium.com/melonport-blog/oyente-beta-release-announcement-dd573cac1dff. [Consulté le: 28-sept-2017].

[34] C. F. Torres et M. Steichen, « The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts », *ArXiv190206976 Cs*, févr. 2019.

[35] C. F. Torres, J. Schütte, et R. State, « Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts », dans *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, p. 664-676.

[36] E. Albert, P. Gordillo, A. Rubio, et I. Sergey, « GASTAP: A Gas Analyzer for Smart Contracts », *ArXiv181110403 Cs*, nov. 2018.

[37] « manticore: Symbolic execution tool », 25-févr-2018. [En ligne]. Disponible à: https://github.com/trailofbits/manticore. [Consulté le: 25-févr-2018].

[38] « Securify - Formal Verification of Ethereum Smart Contracts ». [En ligne]. Disponible à: https://securify.ch/. [Consulté le: 25-févr-2018].

[39] ChainSecurity, « Securify is Now on GitHub », *ChainSecurity*, 28-sept-2018. [En ligne]. Disponible à: https://medium.com/chainsecurity/securify-is-now-on-github-d3bec281eafc. [Consulté le: 06-mai-2019].

[40] « Security Scanner for Ethereum Smart Contracts. », 05-mai-2019. [En ligne]. Disponible à: https://github.com/eth-sri/securify. [Consulté le: 06-mai-2019].

[41] I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, et A. Hobor, « Finding The Greedy, Prodigal, and Suicidal Contracts at Scale », *ArXiv180206038 Cs*, févr. 2018.

[42] M. Bartoletti, S. Carta, T. Cimoli, et R. Saia, « Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact », *ArXiv170303779 Cs*, mars 2017.

[43] « Account Types, Gas, and Transactions — Ethereum Homestead 0.1 documentation ». [En ligne]. Disponible à: http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html? [Consulté le: 05-mars-2018].

[44] E. M. Clarke et J. M. Wing, « Formal methods: State of the art and future directions », *ACM Comput. Surv. CSUR*, vol. 28, nº 4, p. 626–643, 1996.

[45] « Dev Update: Formal Methods », *Ethereum Blog*, 01-sept-2016. [En ligne]. Disponible à: https://blog.ethereum.org/2016/09/01/formal-methods-roadmap/. [Consulté le: 01-févr-2018].

[46] A. Dika, « Ethereum Smart Contracts: Security Vulnerabilities and Security Tools », Master's Thesis, NTNU, 2017.

[47] *Cardano Hub*. [En ligne]. Disponible à: https://www.cardanohub.org/en/home/. [Consulté le: 05-mars-2018].

[48] « EOS.IO | Dawn is here ». [En ligne]. Disponible à: https://eos.io/. [Consulté le: 05-mars-2018].

[49] « TRON Foundation : Capture the future slipping away ». [En ligne]. Disponible à: https://debug.tron.network. [Consulté le: 06-mai-2019].

[50] « Ethereum Project ». [En ligne]. Disponible à: https://www.ethereum.org/. [Consulté le: 05-mars-2018].

[51] M. Bartoletti et L. Pompianu, « An empirical analysis of smart contracts: platforms, applications, and design patterns », *ArXiv Prepr. ArXiv170306322*, 2017.

[52] N. Atzei, M. Bartoletti, et T. Cimoli, « A Survey of Attacks on Ethereum Smart Contracts (SoK) », dans *International Conference on Principles of Security and Trust*, 2017, p. 164–186.

[53] « Ethereum Virtual Machine Opcodes », *Ethereum Virtual Machine Opcodes*. [En ligne]. Disponible à: https://ethervm.io/. [Consulté le: 07-mai-2019].

[54] « mythril: Security analysis tool for Ethereum smart contracts », 25-févr-2018. [En ligne]. Disponible à: https://github.com/ConsenSys/mythril. [Consulté le: 25-févr-2018].

[55] « Truffle Suite | Sweet Tools for Smart Contracts », *Truffle Suite*. [En ligne]. Disponible à: https://trufflesuite.com. [Consulté le: 14-mai-2019].

[56] « Truffle Suite | Ganache », *Truffle Suite*. [En ligne]. Disponible à: https://trufflesuite.com/ganache. [Consulté le: 14-mai-2019].

[57] « Ethereum Contracts with Verified Source Codes ». [En ligne]. Disponible à: https://etherscan.io/contractsVerified. [Consulté le: 03-févr-2018].

[58] P. Hegedűs, « Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts », *Technologies*, vol. 7, nº 1, p. 6, mars 2019.

[59] chicxurug, « Repository for MDPI Technologies journal submission: chicxurug/mdpi-technologies-2018-data », 05-déc-2018. [En ligne]. Disponible à: https://github.com/chicxurug/mdpi-technologies-2018-data. [Consulté le: 17-mai-2019].

[60] R. Tonelli, G. Destefanis, M. Marchesi, et M. Ortu, « Smart Contracts Software Metrics: a First Study », *ArXiv180201517 Cs*, févr. 2018.

[61] S. McKie, « Solidity Learning: Revert(), Assert(), and Require() in Solidity, and the New REVERT Opcode in the… », *Medium*, 27-sept-2017. [En ligne]. Disponible à: https://medium.com/blockchannel/the-use-of-revert-assert-and-require-in-solidity-and-the-new-revert-opcode-in-the-evm-1a3a7990e06e. [Consulté le: 16-sept-2018].

[62] « Releases · ethereum/solidity · GitHub ». [En ligne]. Disponible à: https://github.com/ethereum/solidity/releases? [Consulté le: 03-oct-2019].

[63] « Solidity v0.5.0 Breaking Changes — Solidity 0.5.0 documentation ». [En ligne]. Disponible à: https://solidity.readthedocs.io/en/v0.5.0/050-breaking-changes.html. [Consulté le: 29-sept-2019].

[64] « Security Analysis — Mythril v0.21.12 documentation ». [En ligne]. Disponible à: https://mythril-classic.readthedocs.io/en/master/security-analysis.html#speed-vs-coverage. [Consulté le: 29-sept-2019].

[65] « OpenZeppelin · GitHub ». [En ligne]. Disponible à: https://github.com/OpenZeppelin. [Consulté le: 01-oct-2019].

[66] « openzeppelin-contracts/SafeMath.sol at master · OpenZeppelin/openzeppelin-contracts · GitHub ». [En ligne]. Disponible à: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol. [Consulté le: 01-oct-2019].

Bibliographie

[1]

E. Albert, P. Gordillo, B. Livshits, A. Rubio, and I. Sergey, “EthIR: A Framework for High-Level Analysis of Ethereum Bytecode,” *arXiv:1805.07208 [cs]*, May 2018.

[2]

E. Albert, P. Gordillo, A. Rubio, and I. Sergey, “GASTAP: A Gas Analyzer for Smart Contracts,” *arXiv:1811.10403 [cs]*, Nov. 2018.

[3]

S. Amani, M. Bégel, M. Bortin, and M. Staples, “Towards Verifying Ethereum Smart Contract Bytecode in Isabelle/HOL,” *CPP. ACM. To appear*, 2018.

[4]

A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*, 1st ed. O’Reilly Media, Inc., 2014.

[5]

A. M. Antonopoulos and G. W. Ph.D, *Mastering Ethereum: Building Smart Contracts and DApps*. O’Reilly Media, Inc., 2018.

[6]

N. Atzei, M. Bartoletti, and T. Cimoli, “A Survey of Attacks on Ethereum Smart Contracts (SoK),” in *International Conference on Principles of Security and Trust*, 2017, pp. 164–186.

[7]

X. Bai, Z. Cheng, Z. Duan, and K. Hu, “Formal Modeling and Verification of Smart Contracts,” in *Proceedings of the 2018 7th International Conference on Software and Computer Applications - ICSCA 2018*, Kuantan, Malaysia, 2018, pp. 322–326.

[8]

M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, “Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact,” *arXiv:1703.03779 [cs]*, Mar. 2017.

[9]

M. Bartoletti and L. Pompianu, “An empirical analysis of smart contracts: platforms, applications, and design patterns,” *arXiv preprint arXiv:1703.06322*, 2017.

[10]

K. Bhargavan *et al.*, “Formal Verification of Smart Contracts: Short Paper,” 2016, pp. 91–96.

[11]

L. Brent *et al.*, “Vandal: A Scalable Security Analysis Framework for Smart Contracts,” *arXiv:1809.03981 [cs]*, Sep. 2018.

[12]

V. Buterin, “A next-generation smart contract and decentralized application platform,” *white paper*, 2014.

[13]

J. Chang, B. Gao, H. Xiao, J. Sun, and Z. Yang, “sCompile: Critical Path Identification and Analysis for Smart Contracts,” *arXiv:1808.00624 [cs]*, Aug. 2018.

[14]

D. Chaum, “Blind Signatures for Untraceable Payments,” in *Advances in Cryptology*, Springer, Boston, MA, 1983, pp. 199–203.

[15]

T. Chen *et al.*, “An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks,” in *International Conference on Information Security Practice and Experience*, 2017, pp. 3–24.

[16]

E. M. Clarke and J. M. Wing, “Formal methods: State of the art and future directions,” *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[17]

M. Coblenz, “Obsidian: A Safer Blockchain Programming Language,” in *Proceedings of the 39th International Conference on Software Engineering Companion*, 2017, pp. 97–99.

[18]

W. Dai, “B-Money,” 1998. [Online]. Available: http://www.weidai.com/bmoney.txt.

[19]

M. Dameron, “Beigepaper: An Ethereum Technical Specification,” p. 24.

[20]

K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, “Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab,” in *International Conference on Financial Cryptography and Data Security*, 2016, pp. 79–94.

[21]

A. Dika, “Ethereum Smart Contracts: Security Vulnerabilities and Security Tools,” Master’s Thesis, NTNU, 2017.

[22]

Y. Feng, E. Torlak, and R. Bodik, “Precise Attack Synthesis for Smart Contracts,” Feb. 2019.

[23]

N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, “MadMax: surviving out-of-gas conditions in Ethereum smart contracts,” *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–27, Oct. 2018.

[24]

D. Harz and W. Knottenbelt, “Towards Safer Smart Contracts: A Survey of Languages and Verification Methods,” *arXiv:1809.09805 [cs]*, Sep. 2018.

[25]

P. Hegedűs, “Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts,” *Technologies*, vol. 7, no. 1, p. 6, Mar. 2019.

[26]

E. Hildenbrandt *et al.*, “KEVM: A Complete Semantics of the Ethereum Virtual Machine,” 2017.

[27]

Y. Hirai, *Formal verification of Deed contract in Ethereum name service*. 2016.

[28]

Y. Hirai, “Defining the ethereum virtual machine for interactive theorem provers,” in *International Conference on Financial Cryptography and Data Security*, 2017, pp. 520–535.

[29]

A. Juels, A. Kosba, and E. Shi, “The Ring of Gyges: Investigating the Future of Criminal Smart Contracts,” 2016, pp. 283–295.

[30]

J. C. King, “Symbolic execution and program testing,” *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[31]

J. Krupp and C. Rossow, “TEETHER: Gnawing at Ethereum to Automatically Exploit Smart Contracts,” in *27th {USENIX} Security Symposium*, 2018, pp. 1317–1333.

[32]

A. Li and F. Long, “Detecting Standard Violation Errors in Smart Contracts,” *arXiv:1812.07702 [cs]*, Dec. 2018.

[33]

X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, “A Survey on the security of blockchain systems,” *Future Generation Computer Systems*, Aug. 2017.

[34]

J. Lockhard, C. Purdy, and P. Wilsey, “Formal methods for safety critical system specification,” *2014 IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS), Circuits and Systems (MWSCAS), 2014 IEEE 57th International Midwest Symposium on*, p. 201, 2014.

[35]

L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making Smart Contracts Smarter,” in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[36]

A. Mavridou and A. Laszka, “Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach,” *arXiv preprint arXiv:1711.09327*, 2017.

[37]

A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey, “VeriSolid: Correct-by-Design Smart Contracts for Ethereum,” *arXiv:1901.01292 [cs]*, Jan. 2019.

[38]

P. McCorry, A. Hicks, and S. Meiklejohn, “Smart Contracts for Bribing Miners,” in *International Conference on Financial Cryptography and Data Security*, Berlin, Heidelberg, 2018, pp. 3–18.

[39]

A. Mense and M. Flatscher, “Security Vulnerabilities in Ethereum Smart Contracts,” in *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services - iiWAS2018*, Yogyakarta, Indonesia, 2018, pp. 375–380.

[40]

B. Mueller, “Smashing Ethereum Smart Contracts for Fun and Real Profit,” presented at the 9th Annual HITB Security Conference, 2018, p. 54.

[41]

S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[42]

Z. Nehai and F. Bobot, “Deductive Proof of Ethereum Smart Contracts Using Why3,” *arXiv:1904.11281 [cs]*, Apr. 2019.

[43]

Z. Nehai, P.-Y. Piriou, and F. Daumas, “Model-Checking of Smart Contracts,” in *IEEE International Conference on Internet of Things (iThings)*, 2018, p. 9.

[44]

I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, “Finding The Greedy, Prodigal, and Suicidal Contracts at Scale,” *arXiv:1802.06038 [cs]*, Feb. 2018.

[45]

R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen, “Visual emulation for Ethereum’s virtual machine,” in *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–4.

[46]

R. O’Connor, “Simplicity: A New Language for Blockchains,” *arXiv:1711.03028 [cs]*, pp. 107–120, 2017.

[47]

D. Park, Y. Zhang, M. Saxena, P. Daian, and G. Roşu, “A formal verification tool for Ethereum VM bytecode,” in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, Lake Buena Vista, FL, USA, 2018, pp. 912–915.

[48]

S. Rouhani and R. Deters, “Security, Performance, and Applications of Smart Contracts: A Systematic Survey,” *IEEE Access*, vol. 7, pp. 50759–50779, 2019.

[49]

N. Szabo, “Formalizing and Securing Relationships on Public Networks,” *First Monday*, vol. 2, no. 9, Sep. 1997.

[50]

W. J.-W. Tann, X. J. Han, S. S. Gupta, and Y.-S. Ong, “Towards Safer Smart Contracts: A Sequence Learning Approach to Detecting Security Threats,” *arXiv:1811.06632 [cs]*, Nov. 2018.

[51]

R. Tonelli, G. Destefanis, M. Marchesi, and M. Ortu, “Smart Contracts Software Metrics: a First Study,” *arXiv:1802.01517 [cs]*, Feb. 2018.

[52]

C. F. Torres, J. Schütte, and R. State, “Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts,” in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 664–676.

[53]

C. F. Torres and M. Steichen, “The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts,” *arXiv:1902.06976 [cs]*, Feb. 2019.

[54]

P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, “Securify: Practical Security Analysis of Smart Contracts,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS ’18*, Toronto, Canada, 2018, pp. 67–82.

[55]

G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, 2014.

[56]

Zheng Yang and Hang Lei, “Formal Process Virtual Machine for Smart Contracts Verification,” *International Journal of Performability Engineering*, vol. 14, no. 8, pp. 1726–1734, Aug. 2018.

Annexe I : Quantité d'essences par instruction dans Ethereum

Valeur	Nom de l'instruction	Quantité d'essences
0x00	STOP	0
0x01	ADD	3
0x02	MUL	5
0x03	SUB	3
0x04	DIV	5
0x05	SDIV	5
0x06	MOD	5
0x07	SMOD	5
0x08	ADDMOD	8
0x09	MULMOD	8
0x0a	EXP	FORMULE
0x0b	SIGNEXTEND	5
0x10	LT	3
0x11	GT	3
0x12	SLT	3
0x13	SGT	3
0x14	EQ	3
0x15	ISZERO	3
0x16	AND	3
0x17	OR	3
0x18	XOR	3
0x19	NOT	3
0x1a	BYTE	3
0x20	SHA3	FORMULE
0x30	ADDRESS	2
0x31	BALANCE	400
0x32	ORIGIN	2
0x33	CALLER	2
0x34	CALLVALUE	2
0x35	CALLDATALOAD	3
0x36	CALLDATASIZE	2

0x37	CALLDATACOPY	FORMULE
0x38	CODESIZE	2
0x39	CODECOPY	FORMULE
0x3a	GASPRICE	2
0x3b	EXTCODESIZE	700
0x3c	EXTCODECOPY	FORMULE
0x40	BLOCKHASH	20
0x41	COINBASE	2
0x42	TIMESTAMP	2
0x43	NUMBER	2
0x44	DIFFICULTY	2
0x45	GASLIMIT	2
0x50	POP	2
0x51	MLOAD	3
0x52	MSTORE	3
0x53	MSTORE8	3
0x54	SLOAD	200
0x55	SSTORE	FORMULE
0x56	JUMP	8
0x57	JUMPI	10
0x58	PC	2
0x59	MSIZE	2
0x5a	GAS	2
0x5b	JUMPDEST	1
0x60 -- 0x7f	PUSH*	3
0x80 -- 0x8f	DUP*	3
0x90 -- 0x9f	SWAP*	3
0xa0	LOG0	FORMULE
0xa1	LOG1	FORMULE
0xa2	LOG2	FORMULE
0xa3	LOG3	FORMULE
0xa4	LOG4	FORMULE
0xf0	CREATE	32000
0xf1	CALL	FORMULE
0xf2	CALLCODE	FORMULE
0xf3	RETURN	0
0xf4	DELEGATECALL	FORMULE

0xfd	REVERT	
0xfe	ASSERT_FAIL	NA
0xff	SELFDESTRUCT	FORMULE

Annexe II : Résultats de l'échantillonnage

Rang sélection	Nom	Adresse sur Ethereum	Type	Version Solidity	Lignes
1	Bytes32Lib	0x0fcf5a3884585e14fd2d8d131def18f598dfcc2b	Librairie	Inconnu	37
2	NePay	0x1f0480a66883de97d2b054929252aae8f664c15c	Monnaie	^0.4.22	220
3	SafeMathLibExt	0x2F78032912eF59a990264C4C06937dA4dBF550FD	Librairie	^0.4.6	46
4	EC	0x005aae78c0de67642c728504dc9d264ecb9bb312	Librairie	^0.4.2	207
5	AirDrop	0x20e2bf0fc47e65a3caa5e8e17c5cd730cc556db9	Monnaie	^0.4.24	68
6	EthTranchePricing	0x37a2e0cdcc3dec87042dedfd32daca45d2089ecd	Monnaie	Inconnu	958
7	ChickenFarm	0x246a3CC8430909cB9b4ea2C55064045096656792	Jeu	^0.4.18	139
8	Crowdsale	0x2029f1376726e9b1e59a9eb0c40656f1949e6684	Monnaie	^0.4.18	337
9	lockEtherPay	0x38383c8ef701938d52b98cbc818d923b38473f6d	Monnaie	^0.4.18	109
10	BarterCoin	0x398916719321488633 0c3e4b5535428e168cf6bd	Monnaie	^0.4.11	121
11	NeobitTokens	0x4fe3c4e0c8fcf9ef23b1391cad0205b3e338e013	Monnaie	^0.4.18	221
12	LightYears	0x6a3d166e417f578c6fa6581e529fd407be0f76df	Monnaie	^0.4.16	138
13	DonationWallet	0x7a078d85ff5fb35e2595345c0a96a111f4c793f7	Monnaie	^0.4.23	32
14	COSHATokenCNY	0x44d2ff4a96ba416554951f28d239cad0abd2800d	Monnaie	^0.4.10	155
15	RequestBitcoinNodes Validation	0x60fc18f243656532fce2265a5278d95cb3afa034	Monnaie	^0.4.18	1605
16	SetherToken	0x78b039921e84e726eb72e7b1212bb35504c645ca	Monnaie	Inconnu	255
17	NVISIONCASH	0x643e7c00f3e3e67a635cfbbea615fba047af4f42	Monnaie	^0.4.4	144
18	GdprCrowdsale	0x6102a380791a5edfb04bb3e18ea15812a5b82e71	Monnaie	^0.4.19	840
19	ExploreCoin	0x78219a78b338ee760dff8d1f3e086ceb028a7aa5	Monnaie	^0.4.11	153
20	Anaco_Airdrop	0x791107901275227069ae882990a5aaff51cd658a	Monnaie	^0.4.24	136
21	OneMonthLockup	0x8f45471d4a900f91996871fbfbc2e1d076d936c0	Monnaie	^0.4.18	75
22	BancorMarketMaker	0x9fa14ed09da35b21a764e7c7f7724c70a26b6bfb	Monnaie	^0.4.16	63
23	CAVAsset	0x96b2cd70523aefe92e1cb56de72856ea30498547	Librairie	^0.4.11	254
24	EtherScrolls	0x992d6d699d3f7c627a9be1a5f6020a05ecb86200	Jeu	^0.4.19	550
25	MyAdvancedToken	0x8912358D977e123b51EcAd1fFA0cC4A7e32FF774	Monnaie	^0.4.16	244
26	AuctusPreSale Distribution	0xa39ca2768a7b76aa3bcab68c4d4debf9a32c5434	Monnaie	^0.4.21	84
27	CGCGToken	0xaa561f3baafda7336d6830e5de58dc32f176f661	Monnaie	^0.4.18	106
28	GodviewChain	0xb3bc2C53a02B7c09a2A4957B195B202585138fb4	Monnaie	^0.4.8	92
29	LoopringProtocolImpl	0xb1170de31c7f72ab62535862c97f5209e356991b	Protocole	^0.4.21	1254

30	PixoArenaFounder Token	0xbdd021489de3f083e5aaa2d8cb6ba62db2902485	Monnaie	^0.4.18	255
31	KYC	0xc9e8045616abbdf535fda1fdbfe04b4f42101b2e	Monnaie	^0.4.18	180
32	YeedToken	0xca2796f9f61dc7b238aab043971e49c6164df375	Monnaie	0.4.24	355
33	EtherGames	0xd2cbca4449adb54ecddb3a65faf204b5e1790c3e	Jeu	^0.4.18	425
34	Lottery	0xd834bED9042c2004493f8c8f28C5AaAAb1f36780	Loterie	^0.4.11	87
35	MetabaseCrowdSale	0xde0e95cd7572537842045e9d0051a2c3923794c8	Monnaie	^0.4.18	541
36	Tripy	0xe29cd51e9c816181bd51d1ab781c3556205d44ff	Monnaie	^0.4.18	314
37	BarrelAgedFOMO	0xeac97639bd994496c974040536f4e99cbd9d692e	Monnaie	^0.4.18	548
38	MEMESCrowdsale	0xf2cff4826bf8656173084e86ade7b65bf9aeef5b	Monnaie	^0.4.16	69
39	PlaakPreSale	0xf902b3a6daee738ef7d165d8d38a7edb1336fbfe	Monnaie	^0.4.13	111
40	Lottery (code source diffère de Lottery à la position 34)	0xffff0031d8768861b3172a7c7dc7d91b646f53db	Loterie	^0.4.20	208

Annexe III : Résultats de l'analyse par l'inspection manuelle et Sherlock

Rang	Nom	Inspection manuelle	Sherlock PR = 20	Sherlock PR = 50	Sherlock PR = 100
1	Bytes32Lib	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
2	NePay	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
3	SafeMathLibExt	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
4	EC	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
5	AirDrop	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
6	EthTranchePricing	ASSERT_FAIL TIMESTAMP	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
7	ChickenFarm	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
8	Crowdsale	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
9	lockEtherPay	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
10	BarterCoin	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
11	NeobitTokens	Aucun Vol	TIME	TIME	TIME
12	LightYears	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
13	DonationWallet	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
14	COSHATokenCNY	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
15	RequestBitcoinNodes Validation	ASSERT_FAIL	ASSERT_FAIL TIME	ASSERT_FAIL TIME	ASSERT_FAIL TIME
16	SetherToken	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
17	NVISIONCASH	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
18	GdprCrowdsale	ASSERT_FAIL	Aucun Vol	Aucun Vol	Aucun Vol
19	ExploreCoin	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
20	Anaco_Airdrop	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
21	OneMonthLockup	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
22	BancorMarketMaker	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
23	CAVAsset	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
24	EtherScrolls	ASSERT_FAIL	Aucun Vol	Aucun Vol	ASSERT_FAIL
25	MyAdvancedToken	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
26	AuctusPreSale Distribution	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL

27	CGCGToken	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
28	GodviewChain	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
29	LoopringProtocolImpl	Aucun Vol	TIME	TIME	TIME
30	PixoArenaFounder Token	Aucun Vol	TIME	TIME	TIME
31	KYC	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
32	YeedToken	ASSERT_FAIL	Aucun Vol	Aucun Vol	Aucun Vol
33	EtherGames	ASSERT_FAIL	Aucun Vol	ASSERT_FAIL	ASSERT_FAIL
34	Lottery	Aucun Vol	BlockNumber	BlockNumber	BlockNumber
35	MetabaseCrowdSale	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
36	Tripy	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol
37	BarrelAgedFOMO	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
38	MEMESCrowdsale	Aucun Vol	TIME	TIME	TIME
39	PlaakPreSale	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL	ASSERT_FAIL
40	Lottery	Aucun Vol	Aucun Vol	Aucun Vol	Aucun Vol

Annexe IV : Comparaison entre l'inspection manuelle et Sherlock

(PR = 20)

Rang	Nom	Bonne détection - frauduleux	Bonne détection - non frauduleux	Faux positif	Faux négatif
1	Bytes32Lib	X			
2	NePay	X			
3	SafeMathLibExt	X			
4	EC	X			
5	AirDrop		X		
6	EthTranchePricing	X			
7	ChickenFarm	X			
8	Crowdsale	X			
9	lockEtherPay		X		
10	BarterCoin	X			
11	NeobitTokens			X	
12	LightYears		X		
13	DonationWallet		X		
14	COSHATokenCNY		X		
15	RequestBitcoinNodes Validation	X			
16	SetherToken	X			
17	NVISIONCASH		X		
18	GdprCrowdsale				X
19	ExploreCoin	X			
20	Anaco_Airdrop	X			
21	OneMonthLockup		X		
22	BancorMarketMaker	X			
23	CAVAsset		X		
24	EtherScrolls				X
25	MyAdvancedToken	X			
26	AuctusPreSale Distribution	X			
27	CGCGToken		X		

28	GodviewChain		X		
29	LoopringProtocolImpl			X	
30	PixoArenaFounder Token			X	
31	KYC		X		
32	YeedToken				X
33	EtherGames				X
34	Lottery			X	
35	MetabaseCrowdSale		X		
36	Tripy		X		
37	BarrelAgedFOMO	X			
38	MEMESCrowdsale			x	
39	PlaakPreSale	X			
40	Lottery		X		

Annexe V : Comparaison entre l'inspection manuelle et Sherlock

(PR = 50)

Rang	Nom	Bonne détection - frauduleux	Bonne détection - non frauduleux	Faux positif	Faux négatif
1	Bytes32Lib	X			
2	NePay	X			
3	SafeMathLibExt	X			
4	EC	X			
5	AirDrop		X		
6	EthTranchePricing	X			
7	ChickenFarm	X			
8	Crowdsale	X			
9	lockEtherPay		X		
10	BarterCoin	X			
11	NeobitTokens			X	
12	LightYears		X		
13	DonationWallet		X		
14	COSHATokenCNY		X		
15	RequestBitcoinNodes Validation	X			
16	SetherToken	X			
17	NVISIONCASH		X		
18	GdprCrowdsale				X
19	ExploreCoin	X			
20	Anaco_Airdrop	X			
21	OneMonthLockup		X		
22	BancorMarketMaker	X			
23	CAVAsset		X		
24	EtherScrolls				X
25	MyAdvancedToken	X			
26	AuctusPreSale Distribution	X			
27	CGCGToken		X		

28	GodviewChain		X		
29	LoopringProtocolImpl			X	
30	PixoArenaFounder Token			X	
31	KYC		X		
32	YeedToken				X
33	EtherGames				X
34	Lottery			X	
35	MetabaseCrowdSale		X		
36	Tripy		X		
37	BarrelAgedFOMO	X			
38	MEMESCrowdsale			x	
39	PlaakPreSale	X			
40	Lottery		X		

Annexe VI : Comparaison entre l'inspection manuelle et Sherlock

(PR = 100)

Rang	Nom	Bonne détection - frauduleux	Bonne détection – non frauduleux	Faux positif	Faux négatif
1	Bytes32Lib	X			
2	NePay	X			
3	SafeMathLibExt	X			
4	EC	X			
5	AirDrop		X		
6	EthTranchePricing	X			
7	ChickenFarm	X			
8	Crowdsale	X			
9	lockEtherPay		X		
10	BarterCoin	X			
11	NeobitTokens			X	
12	LightYears		X		
13	DonationWallet		X		
14	COSHATokenCNY		X		
15	RequestBitcoinNodes Validation	X			
16	SetherToken	X			
17	NVISIONCASH		X		
18	GdprCrowdsale				X
19	ExploreCoin	X			
20	Anaco_Airdrop	X			
21	OneMonthLockup		X		
22	BancorMarketMaker	X			
23	CAVAsset		X		
24	EtherScrolls	X			
25	MyAdvancedToken	X			
26	AuctusPreSale Distribution	X			
27	CGCGToken		X		

28	GodviewChain		X		
29	LoopringProtocolImpl			X	
30	PixoArenaFounder Token			X	
31	KYC		X		
32	YeedToken				x
33	EtherGames	X			
34	Lottery			X	
35	MetabaseCrowdSale		X		
36	Tripy		X		
37	BarrelAgedFOMO	X			
38	MEMESCrowdsale			X	
39	PlaakPreSale	X			
40	Lottery		X		

Annexe VII : Données de sortie de Sherlock (PR=20)

nom contrat: 1-Bytes32Lib
adresse contrat: 0x0fcf5a3884585e14fd2d8d131def18f598dfcc2b

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 1-Bytes32Lib
Fonction: f75fa5e1
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 3077 Type Vol: Instruction Invalide
  Instruction: RETURN Solvable: True Essence Maximum: 3581
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 3369 Type Vol: Instruction Invalide
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 4805 Type Vol: Instruction Invalide
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 4809 Type Vol: Instruction Invalide
 Nombre total de chemins: 5
  Essence Minimale: 3077
  Essence Maximale: 4809

Fonction: 0dcfb018
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1494 Type Vol: Instruction Invalide
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1899 Type Vol: Instruction Invalide
 Nombre total de chemins: 2
  Essence Minimale: 1494
  Essence Maximale: 1899

Temps exécution: 124.89686155319214
Analyse terminée
---

nom contrat: 2-NePay
adresse contrat: 0x1f0480a66883de97d2b054929252aae8f664c15c

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 2-NePay
Fonction:
  Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 58005 Type Vol: Instruction Invalide
 Nombre total de chemins: 1
  Essence Minimale: 58005
  Essence Maximale: 58005

Fonction: 23b872dd
  Instruction: REVERT Solvable: True Essence Maximum: 613
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1037 Type Vol: Instruction Invalide
  Instruction: REVERT Solvable: True Essence Maximum: 1421
  Instruction: REVERT Solvable: True Essence Maximum: 2461
 Nombre total de chemins: 4
  Essence Minimale: 613
  Essence Maximale: 2461

Fonction: a9059cbb
  Instruction: REVERT Solvable: True Essence Maximum: 789
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1198 Type Vol: Instruction Invalide
  Instruction: REVERT Solvable: True Essence Maximum: 1582

Instruction: REVERT Solvable: True Essence Maximum: 2621
Nombre total de chemins: 4
Essence Minimale: 789
Essence Maximale: 2621

Appel autre contrat détecté
Temps exécution: 50.32502293586731
Analyse terminée
---

nom contrat: 3-SafeMathLibExt
adresse contrat: 0x2F78032912eF59a990264C4C06937dA4dBF550FD

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 3-SafeMathLibExt
Fonction:
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 281 Type Vol: Instruction Invalide
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 368 Type Vol: Instruction Invalide
Nombre total de chemins: 2
Essence Minimale: 281
Essence Maximale: 368

Fonction: f4f3bdc1
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 277 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 635
Nombre total de chemins: 2
Essence Minimale: 277
Essence Maximale: 635

Fonction: fallback
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 222 Type Vol: Instruction Invalide
Nombre total de chemins: 1
Essence Minimale: 222
Essence Maximale: 222

Fonction: 66098d4f
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 242 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 599
Nombre total de chemins: 2
Essence Minimale: 242
Essence Maximale: 599

Fonction: a12f69e0
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 255 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 725
Nombre total de chemins: 2
Essence Minimale: 255
Essence Maximale: 725

Temps exécution: 391.18469524383545
Analyse terminée
---

nom contrat: 4-EC
adresse contrat: 0x005aae78c0de67642c728504dc9d264ecb9bb312

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 4-EC
Fonction: c373d54f
  Instruction: REVERT Solvable: True Essence Maximum: 487
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 934 Type Vol: Instruction Invalide
 Nombre total de chemins: 2
  Essence Minimale: 487
  Essence Maximale: 934

Fonction: cb577480
  Instruction: REVERT Solvable: True Essence Maximum: 509
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1412 Type Vol: Instruction Invalide
 Nombre total de chemins: 2
  Essence Minimale: 509
  Essence Maximale: 1412

Fonction: 47b6fa28
  Instruction: REVERT Solvable: True Essence Maximum: 333
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 780 Type Vol: Instruction Invalide
 Nombre total de chemins: 2
  Essence Minimale: 333
  Essence Maximale: 780

Fonction: 8940aebe
  Instruction: REVERT Solvable: True Essence Maximum: 421
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1334 Type Vol: Instruction Invalide
  Instruction: RETURN Solvable: True Essence Maximum: 969
 Nombre total de chemins: 3
  Essence Minimale: 421
  Essence Maximale: 1334

Temps exécution: 50.381813526153564
Analyse terminée
---

nom contrat: 5-AirDrop
adresse contrat: 0x20e2bf0fc47e65a3caa5e8e17c5cd730cc556db9

Aucun vol d'essences détecté dans le contrat 5-AirDrop
Appel autre contrat détecté
Temps exécution: 13.242922067642212
Analyse terminée
---

nom contrat: 6-EthTranchePricing
adresse contrat: 0x37a2e0cdcc3dec87042dedfd32daca45d2089ecd

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 6-EthTranchePricing
Fonction: 6f079f90
  Instruction: REVERT Solvable: True Essence Maximum: 308
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1235 Type Vol: Instruction Invalide
 Nombre total de chemins: 2
  Essence Minimale: 308
  Essence Maximale: 1235

Fonction: 26c25962

Instruction: REVERT Solvable: True Essence Maximum: 242
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 286 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 1578
Nombre total de chemins: 3
Essence Minimale: 242
Essence Maximale: 1578

Fonction: d972e8ad
Instruction: REVERT Solvable: True Essence Maximum: 418
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 468 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 1805
Nombre total de chemins: 3
Essence Minimale: 418
Essence Maximale: 1805

Temps exécution: 43.02820372581482
Analyse terminée
---

nom contrat: 7-ChickenFarm
adresse contrat: 0x246a3CC8430909cB9b4ea2C55064045096656792

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 7-ChickenFarm
Fonction: 3bc0461a
Instruction: REVERT Solvable: True Essence Maximum: 427
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 629 Type Vol: Instruction Invalide
Nombre total de chemins: 2
Essence Minimale: 427
Essence Maximale: 629

Temps exécution: 60.006446838378906
Analyse terminée
---

nom contrat: 8-Crowdsale
adresse contrat: 0x2029f1376726e9b1e59a9eb0c40656f1949e6684

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 8-Crowdsale
Fonction:
Instruction: REVERT Solvable: True Essence Maximum: 2296
Instruction: CALL Solvable: True Essence Maximum: 2299
Instruction: REVERT Solvable: True Essence Maximum: 37024
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 37402 Type Vol: Instruction Invalide
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 38052 Type Vol: Instruction Invalide
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 38036 Type Vol: Instruction Invalide
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 37986 Type Vol: Instruction Invalide
Nombre total de chemins: 7
Essence Minimale: 2296
Essence Maximale: 38052

Fonction: 0b151811
Instruction: REVERT Solvable: True Essence Maximum: 204
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 724 Type Vol: Instruction Invalide
Nombre total de chemins: 2
Essence Minimale: 204

Essence Maximale: 724

Fonction: 4aa66b28
Instruction: REVERT Solvable: True Essence Maximum: 314
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 951 Type Vol: Instruction Invalide
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 935 Type Vol: Instruction Invalide
Nombre total de chemins: 3
Essence Minimale: 314
Essence Maximale: 951424

Fonction: 65c40b07
Instruction: REVERT Solvable: True Essence Maximum: 380
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 858 Type Vol: Instruction Invalide
Nombre total de chemins: 2
Essence Minimale: 380
Essence Maximale: 858

Appel autre contrat détecté
Temps exécution: 424.5301251411438
Analyse terminée
---

nom contrat: 9-lockEtherPay
adresse contrat: 0x38383c8ef701938d52b98cbc818d923b38473f6d

Aucun vol d'essences détecté dans le contrat 9-lockEtherPay
Appel autre contrat détecté
Temps exécution: 37.629918575286865
Analyse terminée
---

nom contrat: 10-BarterCoin
adresse contrat: 0x3989167193214886330c3e4b5535428e168cf6bd

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 10-BarterCoin
Fonction: a9059cbb
Instruction: REVERT Solvable: True Essence Maximum: 402
Instruction: REVERT Solvable: True Essence Maximum: 528
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1272 Type Vol: Instruction Invalide
Nombre total de chemins: 3
Essence Minimale: 402
Essence Maximale: 1272

Fonction: 23b872dd
Instruction: REVERT Solvable: True Essence Maximum: 270
Instruction: REVERT Solvable: True Essence Maximum: 429
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 2141 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 81714
Nombre total de chemins: 4
Essence Minimale: 270
Essence Maximale: 81714

Temps exécution: 50.26245713233948
Analyse terminée
---

nom contrat: 11-NeobitTokens
adresse contrat: 0x4fe3c4e0c8fcf9ef23b1391cad0205b3e338e013

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 11-NeobitTokens
Fonction:
  Instruction: REVERT Solvable: True Essence Maximum: 1007 Type Vol: Time
  Instruction: CALL Solvable: True Essence Maximum: 56148 Type Vol: Time Time
  Instruction: REVERT Solvable: True Essence Maximum: 90897 Type Vol: Time Time
  Instruction: STOP Solvable: True Essence Maximum: 90886 Type Vol: Time Time
  Instruction: REVERT Solvable: False Essence Maximum: 53865
 Nombre total de chemins: 5
  Essence Minimale: 1007
  Essence Maximale: 90897

Appel autre contrat détecté
Temps exécution: 64.31546092033386
Analyse terminée
---

nom contrat: 12-LightYears
adresse contrat: 0x6a3d166e417f578c6fa6581e529fd407be0f76df

Aucun vol d'essences détecté dans le contrat 12-LightYears
Temps exécution: 67.37521624565125
Analyse terminée
---

nom contrat: 13-DonationWallet
adresse contrat: 0x7a078d85ff5fb35e2595345c0a96a111f4c793f7

Aucun vol d'essences détecté dans le contrat 13-DonationWallet
Appel autre contrat détecté
Temps exécution: 204.98337030410767
Analyse terminée
---

nom contrat: 14-COSHATokenCNY
adresse contrat: 0x44d2ff4a96ba416554951f28d239cad0abd2800d

Aucun vol d'essences détecté dans le contrat 14-COSHATokenCNY
Temps exécution: 86.93266987800598
Analyse terminée
---

nom contrat: 15-RequestBitcoinNodesValidation
adresse contrat: 0x60fc18f243656532fce2265a5278d95cb3afa034

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 15-RequestBitcoinNodesValidation
Fonction:
  Instruction: REVERT Solvable: True Essence Maximum: 141
  Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 825 Type Vol: Instruction Invalide
  Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 11997 Type Vol: Instruction Invalide Time
  Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12275 Type Vol: Instruction Invalide Time
  Instruction: REVERT Solvable: False Essence Maximum: 12508 Type Vol: Time

Instruction: REVERT Solvable: False Essence Maximum: 12386 Type Vol: Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12279 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12557 Type Vol: Instruction Invalide Time
Instruction: REVERT Solvable: False Essence Maximum: 12668 Type Vol: Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12561 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12839 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12843 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 13125 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12282 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12560 Type Vol: Instruction Invalide Time
Instruction: REVERT Solvable: False Essence Maximum: 12671 Type Vol: Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12564 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12842 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12846 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 13128 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12567 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12845 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12849 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 13131 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 12852 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 13134 Type Vol: Instruction Invalide Time
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 13137 Type Vol: Instruction Invalide Time
Instruction: REVERT Solvable: False Essence Maximum: 248
Instruction: RETURN Solvable: False Essence Maximum: 1006
Instruction: REVERT Solvable: False Essence Maximum: 270
Instruction: REVERT Solvable: False Essence Maximum: 6204
Instruction: CALL Solvable: False Essence Maximum: 6207
Instruction: REVERT Solvable: False Essence Maximum: 40932
Instruction: REVERT Solvable: False Essence Maximum: 41170
Instruction: CALL Solvable: False Essence Maximum: 43393
Instruction: REVERT Solvable: False Essence Maximum: 78118
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 78347 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: False Essence Maximum: 292
Instruction: REVERT Solvable: False Essence Maximum: 1113
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 1453 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: False Essence Maximum: 1561
Instruction: REVERT Solvable: False Essence Maximum: 1586
Instruction: REVERT Solvable: False Essence Maximum: 1606
Instruction: REVERT Solvable: False Essence Maximum: 10999
Instruction: REVERT Solvable: False Essence Maximum: 336
Instruction: REVERT Solvable: False Essence Maximum: 1197
Instruction: REVERT Solvable: False Essence Maximum: 358
Instruction: RETURN Solvable: False Essence Maximum: 1494
Instruction: REVERT Solvable: False Essence Maximum: 380
Instruction: RETURN Solvable: False Essence Maximum: 1516
Instruction: REVERT Solvable: False Essence Maximum: 402
Instruction: RETURN Solvable: False Essence Maximum: 1160
Instruction: REVERT Solvable: False Essence Maximum: 424
Instruction: REVERT Solvable: False Essence Maximum: 3479
Instruction: CALL Solvable: False Essence Maximum: 3482
Instruction: REVERT Solvable: False Essence Maximum: 38207
Instruction: REVERT Solvable: False Essence Maximum: 446
Instruction: CALL Solvable: False Essence Maximum: 2443
Instruction: REVERT Solvable: False Essence Maximum: 468
Instruction: REVERT Solvable: False Essence Maximum: 6281

Instruction: CALL Solvable: False Essence Maximum: 6284
Instruction: REVERT Solvable: False Essence Maximum: 41009
Instruction: REVERT Solvable: False Essence Maximum: 490
Instruction: REVERT Solvable: False Essence Maximum: 1311
Instruction: REVERT Solvable: False Essence Maximum: 512
Instruction: RETURN Solvable: False Essence Maximum: 1648
Instruction: REVERT Solvable: False Essence Maximum: 534
Instruction: REVERT Solvable: False Essence Maximum: 556
Nombre total de chemins: 68
Essence Minimale: 141
Essence Maximale: 78347

Fonction: 1ef30ff5
Instruction: REVERT Solvable: True Essence Maximum: 1352 Type Vol: Time
Nombre total de chemins: 1
Essence Minimale: 1352
Essence Maximale: 1352

Appel autre contrat détecté
Temps exécution: 1121.5337572097778
Analyse terminée
---

nom contrat: 16-SetherToken
adresse contrat: 0x78b039921e84e726eb72e7b1212bb35504c645ca

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 16-SetherToken
Fonction: a9059cbb
Instruction: REVERT Solvable: True Essence Maximum: 465
Instruction: REVERT Solvable: True Essence Maximum: 1228
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 2294 Type Vol: Instruction Invalide
Nombre total de chemins: 3
Essence Minimale: 465
Essence Maximale: 2294

Fonction: 23b872dd
Instruction: REVERT Solvable: True Essence Maximum: 289
Instruction: REVERT Solvable: True Essence Maximum: 1070
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 3074 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 83414
Nombre total de chemins: 4
Essence Minimale: 289
Essence Maximale: 83414

Temps exécution: 59.350902795791626
Analyse terminée
---

nom contrat: 17-NVISIONCASH
adresse contrat: 0x643e7c00f3e3e67a635cfbbea615fba047af4f42

Aucun vol d'essences détecté dans le contrat 17-NVISIONCASH
Appel autre contrat détecté
Temps exécution: 106.8192846775055
Analyse terminée

---

nom contrat: 18-GdprCrowdsale
adresse contrat: 0x6102a380791a5edfb04bb3e18ea15812a5b82e71

Aucun vol d'essences détecté dans le contrat 18-GdprCrowdsale
Temps exécution: 47.13301706314087
Analyse terminée
---

nom contrat: 19-ExploreCoin
adresse contrat: 0x78219a78b338ee760dff8d1f3e086ceb028a7aa5

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 19-ExploreCoin
Fonction: dd62ed3e
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 459 Type Vol: Instruction Invalide
  Instruction: RETURN Solvable: True Essence Maximum: 1869
 Nombre total de chemins: 2
  Essence Minimale: 459
  Essence Maximale: 1869

Fonction: 06fdde03
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 195 Type Vol: Instruction Invalide
  Instruction: RETURN Solvable: True Essence Maximum: 2271
 Nombre total de chemins: 2
  Essence Minimale: 195
  Essence Maximale: 2271

Fonction: 18160ddd
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 239 Type Vol: Instruction Invalide
  Instruction: RETURN Solvable: True Essence Maximum: 1012
 Nombre total de chemins: 2
  Essence Minimale: 239
  Essence Maximale: 1012

Fonction: f2fde38b
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 481 Type Vol: Instruction Invalide
  Instruction: REVERT Solvable: True Essence Maximum: 1348
 Nombre total de chemins: 2
  Essence Minimale: 481
  Essence Maximale: 1348

Fonction: 23b872dd
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 261 Type Vol: Instruction Invalide
  Instruction: REVERT Solvable: True Essence Maximum: 427
  Instruction: RETURN Solvable: True Essence Maximum: 1580
  Instruction: RETURN Solvable: True Essence Maximum: 2564
 Nombre total de chemins: 4
  Essence Minimale: 261
  Essence Maximale: 2564

Fonction: 313ce567
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 283 Type Vol: Instruction Invalide
  Instruction: RETURN Solvable: True Essence Maximum: 1047
 Nombre total de chemins: 2

Essence Minimale: 283
Essence Maximale: 1047

Fonction: 95d89b41
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 371 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 2447
Nombre total de chemins: 2
Essence Minimale: 371
Essence Maximale: 2447

Fonction: a9059cbb
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 393 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: True Essence Maximum: 529
Instruction: RETURN Solvable: True Essence Maximum: 1659
Instruction: RETURN Solvable: True Essence Maximum: 1670
Nombre total de chemins: 4
Essence Minimale: 393
Essence Maximale: 1670

Fonction: 5ebdd159
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 327 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 1617
Nombre total de chemins: 2
Essence Minimale: 327
Essence Maximale: 1617

Fonction: b387ef92
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 415 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: True Essence Maximum: 1245
Nombre total de chemins: 2
Essence Minimale: 415
Essence Maximale: 1245

Fonction: 70a08231
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 349 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 1447
Nombre total de chemins: 2
Essence Minimale: 349
Essence Maximale: 1447

Fonction: c763400e
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 437 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: True Essence Maximum: 1304
Nombre total de chemins: 2
Essence Minimale: 437
Essence Maximale: 1304

Fonction: 095ea7b3
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 217 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 28355
Nombre total de chemins: 2
Essence Minimale: 217
Essence Maximale: 28355

Fonction: 519ee19e

Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 305 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 1069
Nombre total de chemins: 2
Essence Minimale: 305
Essence Maximale: 1069

Temps exécution: 48.48213744163513
Analyse terminée
---

nom contrat: 20-Anaco_Airdrop
adresse contrat: 0x791107901275227069ae882990a5aaff51cd658a

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 20-Anaco_Airdrop
Fonction:
Instruction: REVERT Solvable: True Essence Maximum: 162
Instruction: REVERT Solvable: True Essence Maximum: 165
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 885 Type Vol: Instruction Invalide
Nombre total de chemins: 3
Essence Minimale: 162
Essence Maximale: 885

Temps exécution: 134.05529618263245
Analyse terminée
---

nom contrat: 21-OneMonthLockup
adresse contrat: 0x8f45471d4a900f91996871fbfbc2e1d076d936c0

Aucun vol d'essences détecté dans le contrat 21-OneMonthLockup
Appel autre contrat détecté
Temps exécution: 17.79272437095642
Analyse terminée
---

nom contrat: 22-BancorMarketMaker
adresse contrat: 0x9fa14ed09da35b21a764e7c7f7724c70a26b6bfb

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 22-BancorMarketMaker
Fonction: 325add98
Instruction: REVERT Solvable: True Essence Maximum: 563
Instruction: REVERT Solvable: True Essence Maximum: 962
Instruction: REVERT Solvable: True Essence Maximum: 3308
Instruction: CALL Solvable: True Essence Maximum: 3311
Instruction: REVERT Solvable: True Essence Maximum: 38036
Instruction: CALL Solvable: True Essence Maximum: 40452
Instruction: REVERT Solvable: True Essence Maximum: 75177
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 75528 Type Vol: Instruction Invalide
Nombre total de chemins: 8
Essence Minimale: 563
Essence Maximale: 75528

Fonction: abbef24e
Instruction: REVERT Solvable: True Essence Maximum: 1002
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1124 Type Vol: Instruction Invalide

Nombre total de chemins: 2
Essence Minimale: 1002
Essence Maximale: 1124

Appel autre contrat détecté
Temps exécution: 28.426782846450806
Analyse terminée
---

nom contrat: 23-CAVAsset
adresse contrat: 0x96b2cd70523aefe92e1cb56de72856ea30498547

Aucun vol d'essences détecté dans le contrat 23-CAVAsset
Appel autre contrat détecté
Temps exécution: 78.27175211906433
Analyse terminée
---

nom contrat: 24-EtherScrolls
adresse contrat: 0x992d6d699d3f7c627a9be1a5f6020a05ecb86200

Aucun vol d'essences détecté dans le contrat 24-EtherScrolls
Appel autre contrat détecté
Temps exécution: 51.30128860473633
Analyse terminée
---
nom contrat: 25-MyAdvancedToken
adresse contrat: 0x8912358D977e123b51EcAd1fFA0cC4A7e32FF774

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 25-MyAdvancedToken
Fonction: a6f2ae3a
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 927 Type Vol: Instruction Invalide
Nombre total de chemins: 1
Essence Minimale: 927
Essence Maximale: 927

Temps exécution: 66.95342445373535
Analyse terminée
---

nom contrat: 26-AuctusPreSaleDistribution
adresse contrat: 0xa39ca2768a7b76aa3bcab68c4d4debf9a32c5434

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 26-AuctusPreSaleDistribution
Fonction: 1223f290
Instruction: REVERT Solvable: True Essence Maximum: 544
Instruction: REVERT Solvable: True Essence Maximum: 4615
Instruction: CALL Solvable: True Essence Maximum: 4612
Instruction: REVERT Solvable: True Essence Maximum: 39337
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 66064 Type Vol: Instruction Invalide
Nombre total de chemins: 5
Essence Minimale: 544
Essence Maximale: 66064

Appel autre contrat détecté

Temps exécution: 36.538976430892944
Analyse terminée
---

nom contrat: 27-CGCGToken
adresse contrat: 0xaa561f3baafda7336d6830e5de58dc32f176f661

Aucun vol d'essences détecté dans le contrat 27-CGCGToken
Temps exécution: 39.209991216659546
Analyse terminée
---

nom contrat: 28-GodviewChain
adresse contrat: 0xb3bc2C53a02B7c09a2A4957B195B202585138fb4

Aucun vol d'essences détecté dans le contrat 28-GodviewChain
Temps exécution: 64.20478582382202
Analyse terminée
---

nom contrat: 29-LoopringProtocolImpl
adresse contrat: 0xb1170de31c7f72ab62535862c97f5209e356991b

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 29-LoopringProtocolImpl
Fonction: 8865cbd6
  Instruction: REVERT Solvable: True Essence Maximum: 698
  Instruction: REVERT Solvable: True Essence Maximum: 3203
  Instruction: CALL Solvable: True Essence Maximum: 3200
  Instruction: REVERT Solvable: True Essence Maximum: 37925
  Instruction: REVERT Solvable: True Essence Maximum: 38157 Type Vol: Time
  Instruction: CALL Solvable: True Essence Maximum: 40159 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 74884 Type Vol: Time
  Instruction: STOP Solvable: True Essence Maximum: 77575 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 3216 Type Vol: Time
  Instruction: CALL Solvable: True Essence Maximum: 3213 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 37938 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 38170 Type Vol: Time Time
  Instruction: CALL Solvable: True Essence Maximum: 40172 Type Vol: Time Time
  Instruction: REVERT Solvable: True Essence Maximum: 74897 Type Vol: Time Time
  Instruction: STOP Solvable: True Essence Maximum: 77588 Type Vol: Time Time
  Instruction: REVERT Solvable: True Essence Maximum: 3227 Type Vol: Time
  Instruction: CALL Solvable: True Essence Maximum: 3224 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 37949 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 38181 Type Vol: Time
  Instruction: CALL Solvable: True Essence Maximum: 40183 Type Vol: Time
  Instruction: REVERT Solvable: True Essence Maximum: 74908 Type Vol: Time
 Nombre total de chemins: 21
  Essence Minimale: 698
  Essence Maximale: 77588

Fonction: bd545f53
  Instruction: REVERT Solvable: True Essence Maximum: 764
  Instruction: REVERT Solvable: True Essence Maximum: 2738
  Instruction: CALL Solvable: True Essence Maximum: 2735
  Instruction: REVERT Solvable: True Essence Maximum: 37460

Instruction: REVERT Solvable: True Essence Maximum: 37692 Type Vol: Time
Instruction: REVERT Solvable: True Essence Maximum: 2751 Type Vol: Time
Instruction: CALL Solvable: True Essence Maximum: 2748 Type Vol: Time
Instruction: REVERT Solvable: True Essence Maximum: 37473 Type Vol: Time
Instruction: REVERT Solvable: True Essence Maximum: 37705 Type Vol: Time Time
Instruction: REVERT Solvable: True Essence Maximum: 2762 Type Vol: Time
Instruction: CALL Solvable: True Essence Maximum: 2759 Type Vol: Time
Instruction: REVERT Solvable: True Essence Maximum: 37484 Type Vol: Time
Nombre total de chemins: 12
Essence Minimale: 764
Essence Maximale: 37705

Appel autre contrat détecté
Temps exécution: 184.8551321029663
Analyse terminée
---

nom contrat: 30-PixoArenaFounderToken
adresse contrat: 0xbdd021489de3f083e5aaa2d8cb6ba62db2902485

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 30-PixoArenaFounderToken
Fonction:
Instruction: REVERT Solvable: True Essence Maximum: 1007 Type Vol: Time
Instruction: REVERT Solvable: True Essence Maximum: 1038 Type Vol: Time
Instruction: STOP Solvable: True Essence Maximum: 55162 Type Vol: Time Time
Instruction: REVERT Solvable: False Essence Maximum: 53887
Nombre total de chemins: 4
Essence Minimale: 1007
Essence Maximale: 55162

Temps exécution: 49.498276472091675
Analyse terminée
---

nom contrat: 31-KYC
adresse contrat: 0xc9e8045616abbdf535fda1fdbfe04b4f42101b2e

Aucun vol d'essences détecté dans le contrat 31-KYC
Temps exécution: 30.372233629226685
Analyse terminée
---

nom contrat: 32-YeedToken
adresse contrat: 0xca2796f9f61dc7b238aab043971e49c6164df375

Aucun vol d'essences détecté dans le contrat 32-YeedToken
Temps exécution: 44.33206629753113
Analyse terminée
---

nom contrat: 33-EtherGames
adresse contrat: 0xd2cbca4449adb54ecddb3a65faf204b5e1790c3e

Aucun vol d'essences détecté dans le contrat 33-EtherGames
Temps exécution: 44.37218976020813

Analyse terminée
---

nom contrat: 34-Lottery
adresse contrat: 0xd834bED9042c2004493f8c8f28C5AaAAb1f36780

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 34-Lottery
Fonction:
  Instruction: REVERT Solvable: True Essence Maximum: 568
  Instruction: REVERT Solvable: True Essence Maximum: 57546 Type Vol: BlockNumber
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 58378 Type Vol: Instruction Invalide BlockNumber
 Nombre total de chemins: 3
  Essence Minimale: 568
  Essence Maximale: 58378

Fonction: fallback
  Instruction: REVERT Solvable: True Essence Maximum: 745
  Instruction: REVERT Solvable: True Essence Maximum: 57723 Type Vol: BlockNumber
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 58555 Type Vol: Instruction Invalide BlockNumber
 Nombre total de chemins: 3
  Essence Minimale: 745
  Essence Maximale: 58555

Temps exécution: 19.53057050704956
Analyse terminée
---

nom contrat: 35-MetabaseCrowdSale
adresse contrat: 0xde0e95cd7572537842045e9d0051a2c3923794c8

Aucun vol d'essences détecté dans le contrat 35-MetabaseCrowdSale
Temps exécution: 25.94014620780945
Analyse terminée
---

nom contrat: 36-Tripy
adresse contrat: 0xe29cd51e9c816181bd51d1ab781c3556205d44ff

Aucun vol d'essences détecté dans le contrat 36-Tripy
Temps exécution: 50.29711103439331
Analyse terminée
---

nom contrat: 37-BarrelAgedFOMO
adresse contrat: 0xeac97639bd994496c974040536f4e99cbd9d692e

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 37-BarrelAgedFOMO
Fonction:
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 995 Type Vol: Instruction Invalide
 Nombre total de chemins: 1
  Essence Minimale: 995
  Essence Maximale: 995

Fonction: fallback
  Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 1348 Type Vol: Instruction Invalide

Nombre total de chemins: 1
Essence Minimale: 1348
Essence Maximale: 1348

Fonction: a391c15b
Instruction: REVERT Solvable: True Essence Maximum: 424
Instruction: ASSERT_FAIL Solvable: True Essence Maximum: 526 Type Vol: Instruction Invalide
Instruction: RETURN Solvable: True Essence Maximum: 899
Nombre total de chemins: 3
Essence Minimale: 424
Essence Maximale: 899

Temps exécution: 47.712838888168335
Analyse terminée
---

nom contrat: 38-MEMESCrowdsale
adresse contrat: 0xf2cff4826bf8656173084e86ade7b65bf9aeef5b

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 38-MEMESCrowdsale
Fonction:
Instruction: REVERT Solvable: True Essence Maximum: 168
Instruction: REVERT Solvable: True Essence Maximum: 590 Type Vol: Time
Instruction: REVERT Solvable: True Essence Maximum: 1012 Type Vol: Time
Nombre total de chemins: 3
Essence Minimale: 168
Essence Maximale: 1012

Fonction: fallback
Instruction: REVERT Solvable: True Essence Maximum: 817
Instruction: REVERT Solvable: True Essence Maximum: 1661 Type Vol: Time
Nombre total de chemins: 2
Essence Minimale: 817
Essence Maximale: 1661

Fonction: 04048001
Instruction: REVERT Solvable: True Essence Maximum: 1239 Type Vol: Time
Nombre total de chemins: 1
Essence Minimale: 1239
Essence Maximale: 1239

Temps exécution: 36.567360639572144
Analyse terminée
---

nom contrat: 39-PlaakPreSale
adresse contrat: 0xf902b3a6daee738ef7d165d8d38a7edb1336fbfe

Bifurcations frauduleuses dans les fonctions ci-dessous dans le contrat 39-PlaakPreSale
Fonction:
Instruction: REVERT Solvable: True Essence Maximum: 994
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 26662 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: False Essence Maximum: 29011
Instruction: CALL Solvable: True Essence Maximum: 29014
Instruction: REVERT Solvable: False Essence Maximum: 63739

Instruction: STOP Solvable: False Essence Maximum: 66367
Instruction: ASSERT_FAIL Solvable: False Essence Maximum: 26877 Type Vol: Instruction Invalide
Instruction: REVERT Solvable: False Essence Maximum: 358
Instruction: REVERT Solvable: False Essence Maximum: 1213
Instruction: REVERT Solvable: False Essence Maximum: 380
Instruction: RETURN Solvable: False Essence Maximum: 1516
Instruction: REVERT Solvable: False Essence Maximum: 402
Instruction: RETURN Solvable: False Essence Maximum: 1538
Instruction: REVERT Solvable: False Essence Maximum: 1237
Nombre total de chemins: 14
Essence Minimale: 358
Essence Maximale: 66367

Appel autre contrat détecté
Temps exécution: 3097.355877161026
Analyse terminée
---

nom contrat: 40-Lottery2
adresse contrat: 0xffff0031d8768861b3172a7c7dc7d91b646f53db

Aucun vol d'essences détecté dans le contrat 40-Lottery2
Appel autre contrat détecté
Temps exécution: 806.4873373508453
Analyse terminée
