Computer Security Symposium 2024
22-25 October 2024

Ethereum スマートコントラクトにおけるバックドア攻撃の実態調査

矢内 直人[1]　西田 直央[1]　海上 勇二[1]

概要：Ethereum スマートコントラクトにおけるバックドア攻撃は, スマートコントラクトの開発者が特権ユーザとしての権限を用いることで, そのスマートコントラクトから生成された資産を不正に操作する攻撃である. バックドア攻撃の対策ツールはこれまでにも検討されているが, 実際にバックドア攻撃が Ethereum スマートコントラクトでどの程度の脅威であるかは, 著者の知る限りこれまで検討されてこなかった. 本稿では Ethereum スマートコントラクトにおいてバックドア攻撃がどれだけ行われているか, Ethereum の全スマートコントラクト 66,283,849 件を収集・解析することで実態調査する. 本稿で明らかにした知見は以下のとおりである. まず今回調査した範囲において, 34,983 件のバックドアを発見した. とくに最も利用されているコンパイラバージョンである v0.4 において, 42.2% のスマートコントラクトがバックドアであることを確認した. 最も作成されていたバックドアは暗号資産を任意のアドレスに転送する ArbitraryTransfer であり, 24,793 件のスマートコントラクトが確認された. また, バックドアには他のバックドアと組み合わせることで攻撃するものもあり, 特に暗号資産の転送を制限する DisableTransfer は 99.9%以上が他のバックドアとの組み合わせだった. バックドアの種類としては, 上記の ArbitraryTransfer と DisableTransfer を組み合わせたものが 8,348 件と最も多いことも確認している. これらの理由については, 攻撃者が転送を任意に制御することで, 自らの資産を増やす手法が多いことを示唆している.

キーワード：Ethereum スマートコントラクト, バックドア攻撃, 実態調査, コンパイラバージョン

An Empirical Study of Backdoor Attacks on Ethereum Smart Contracts

Naoto Yanai[1]　Naohisa Nishida[1]　Yuji Unagami[1]

Abstract: Backdoor attacks on Ethereum smart contracts are attacks whereby a developer of smart contracts maliciously obtains assets generated from the smart contracts by leveraging privilege. Although several detection tools for the backdoor attacks have been proposed, to the best of our knowledge, no empirical study of the backdoor attacks has been conducted so far. In this paper, we conduct an empirical study for backdoor attacks on Ethereum smart contracts by collecting and analyzing 66,283,849 contracts, which are all of the smart contracts that have been generated until now. We then found the following insights. First, we found 34,983 backdoors in the current investigation range. Notably, for the compiler version v0.4, which is the most popular version, 42.2% of the smart contracts are backdoors. The most generated backdoors are ArbitaryTransfer which transfers assets into any address and there are 27,793 smart contracts for the attack. We also found backdoors which can be combined with other backdoors. The most generated such backdoors are DisableTransfer to restricts transactions, and we found that 99.9% of these backdoors are utilized for combinations with the other backdoors. We identify that there are 8,348 combinations of these backdoors, which are significant in the current results. The reason is that an adversary can maliciously obtain assets by arbitrarily controlling transfers of them.

Keywords: searchable encryption, promotion of understanding cryptography, questionnaire study, educational materials

© 2024 Information Processing Society of Japan

This paper is work in progress and not peer-reviewed.

1. 序論

Ethereum スマートコントラクト [22] はブロックチェーンを通じてプログラムを提供できるプラットフォームとして, 近年では分散アプリケーションの開発やトークンを介した資産の取引など, 様々な場面で利用されている. 大まかには, スマートコントラクトはプログラムコードをブロックチェーンにおける P2P ネットワークに展開することで, 自動的なプログラム実行や電子取引が可能になる. （本稿では Ethereum スマートコントラクトはスマートコントラクトと単に呼称する.）

スマートコントラクトにおける安全性上の懸念は様々に示されているが [24], バックドア攻撃（Rug Pull とも呼ばれる）が近年に注目されている [21]. バックドア攻撃は, スマートコントラクトの開発者がその特権を悪用し, スマートコントラクトから生成された通貨やトークンなど暗号資産を不正に操作する攻撃である. 例えば 2018 年 6 月オーストラリアでは 6.6 百万$ がバックドア攻撃により SoarCoin から盗まれた[*1]. このため, スマートコントラクトのバックドア攻撃について, 様々な検討が行われてきた [9, 13, 14].

しかしながら, 著者の知る限り, スマートコントラクトのバックドア攻撃に関する実態調査はこれまでに行われてこなかった. ここでいう実態調査とは攻撃が現実世界でどの程度起きているのか, データを収集・分析することを意味する. 攻撃の実態調査は一般に, その被害規模や影響, 対策を検討する上で重要な意味がある. また, 現実世界での影響を明らかにすることで, 後続の研究の指針を定めることも期待できる. 上述した背景に基づいて, 本稿ではスマートコントラクトのバックドア攻撃について, 以下の問いを明らかにする:

RQ1: バックドア攻撃はどの程度起きているのか？

RQ2: バックドア攻撃で最も多い手法はどのようなものか？

本稿では, 上述した問いに対し, Ethereum の全スマートコントラクトを収集し, バックドア攻撃の有無を調査することで, 現実世界における影響を明らかにする. 具体的には, Etherscan[*2] から 2024 年 5 月末までに作成されたソースコードがあるスマートコントラクト 66,283,849 件を収集し, これらのスマートコントラクトに代表的なバックドア攻撃検知ツール [13,14] を用いることで, 分析調査を行った.

コンパイラバージョン v0.1 から最も利用頻度が高いバージョンである v0.4 [10] までの分析結果として, 合計で 34,983 件のスマートコントラクトがバックドア攻撃を目的として作られていることを確認した. （以降では該当のスマートコントラクトをバックドアコントラクトと呼称する.）とくにコンパイラバージョン v0.4 では当該バージョンで作成されたスマートコントラクトのうち, 42.2%がバックドアコントラクトだった. バックドアコントラクトの中では, ArbitaryTransfer という暗号資産を任意の送信先に転送する手法が最も多く, 24,793 件のバックドアコントラクトを発見した. これは暗号資産を攻撃者に転送することで摂取するという意味で, 攻撃者が単純に利益を得やすいことがその背景にあると考えられる. また, バックドア攻撃は複数の手法を組み合わせているものが多いことも確認している. とくに, 暗号資産の転送を制限する DisableTransfer と呼ばれるは 99.9%以上が組み合わせに用いられており, これは他のバックドアとの組み合わせを前提としていることが示された. とくに多かった組み合わせとして, 上述した ArbitraryTransfer および暗号資産を任意に生成する GenerateToken との組み合わせが 8,348 件と, 全バックドアコントラクトの中でも最も多かった. これは攻撃者が攻撃対象となるユーザの暗号資産の転送を制限すると同時に, その暗号資産を自らに転送あるいは自ら生成取得することで窃取するという利点をもたらしている.

本稿で明らかにした知見を以下に要約する:

- 本稿の調査では 34,983 件のバックドアコントラクトを確認した.
- v0.4 では 42.3%のスマートコントラクトがバックドアコントラクトだった.
- 暗号資産を任意のアドレスに転送する ArbitraryTransfer が 24,793 件と最も多い.
- バックドアコントラクトには他のバックドアとの組み合わせているものが多く, とくに暗号資産の転送を制限する DisableTransfer は 99.9%以上が組み合わせに用いられている.
- 最も多かったバックドアコントラクトは DisableTransfer, ArbitraryTransfer, および暗号資産を任意に生成する GenerateToken の組み合わせであり, 8,348 件あった.

2. 背景

本章では，Ethereum スマートコントラクトの技術的背景と，本研究におけるその関連研究について述べる.

2.1 Ethereum スマートコントラクト

Ethereum では，Solidity などの高級言語によってスマートコントラクトのソースコードが実装され, 現在までは Solidity は v0.1 から v0.8 まで 8 種類のコンパイラがある．ブロックチェーン上にデプロイされたコントラクトは，P2P ネットワーク上のピアと呼ばれるノードによって管理される．スマートコントラクトには，識別子であるコントラクトアドレスが割り振られ，それを用いて Ethereum

[1] パナソニックホールディングス株式会社, Panasonic Holdings Corporation.

[*1] https://nz.finance.yahoo.com/news/backdoor-flaw-sees-australian-firm-115323212.html

[*2] https://etherscan.io/

© 2024 Information Processing Society of Japan

の通貨である Ether の受け取りや関数の実行が行われる．コントラクトはブロックチェーンに記録されるトランザクションの宛先として，そのコントラクトアドレスが指定された場合に呼び出され，トランザクションに含まれる関数や引数などの情報を元にピアが実行環境 Ethereum Virtual Machine 上でコントラクトを実行する．また，ピアによる実行に対する動機付けとして，Ethereum には Ether によって支払われる**ガス**と呼ばれる実行手数料が導入されている．近年のスマートコントラクトは暗号通貨の利用だけではなく，暗号通貨をブロックチェーンの内外にある資産と紐づけたトークンとして管理する機能も頻繁に利用されている [3]．スマートコントラクトへの攻撃は，The DAO のように甚大な金銭的被害を発生させる場合がある．そのため，様々な攻撃対策が研究されている [4]．

2.2 スマートコントラクトのバックドア攻撃

スマートコントラクトのバックドア攻撃について定義する．以下の定義は Ma ら [14] に従う．大まかには，スマートコントラクトのバックドア攻撃により，例えばバックドアコントラクトから生成された暗号資産を持つユーザは，その暗号資産を合意なく凍結される可能性などが起こりえる．

定義 1. 以下の二つの要件を満たすスマートコントラクトをバックドアコントラクトという: コントラクト開発者など特権的な権限を持つユーザのみ実行可能である; 他のユーザが持つトークンなど暗号資産に（特に金銭的な）影響を与える．スマートコントラクトのバックドア攻撃とは，上述したバックドアコントラクトに従う任意の実行を表す．

バックドアコントラクトは既存研究 [14] によると，任意アドレスに存在する暗号資産を破壊する DestroyToken，暗号資産を任意アドレスに転送させる ArbitraryTransfer，トークンを勝手に生成する GenerateToken，暗号資産の転送を制限する DisableTransfer，アカウント自体を凍結させる FrozenAccount がある．これらの詳細は紙面の都合上割愛するが，本稿では上述した 5 つのバックドアコントラクトを調査する．

2.3 関連研究

バックドア攻撃の対策は様々な手法で提案されている．例えば機械学習 [7, 15, 16, 23, 25]，ファジング [2, 14]，情報フロー解析 [13]，シンボリック実行 [9] がある．本稿の調査はツールが公開されているものとして，Pied-Pier [14] と CRPWarner [13] を用いる．他にもさらにツールを導入することで，例えば偽陰性の削減は期待できる [17]．

本稿に最も近い既存研究はバックドアコントラクトの調査である．まず Sharma ら [20] はバックドアコントラクトにおけるトランザクションなど挙動から調査を行った．次に Kaur ら [12] は既存文献と専門家へのアンケートを基に，暗号資産のリスクを調査している．本稿ではスマートコントラクトのソースコードから分析を行う点が，これらの研究とは異なる．

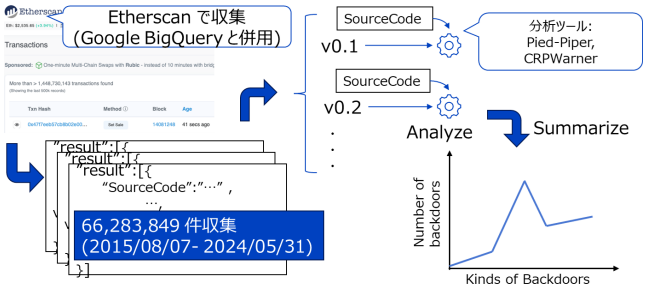

図 1 全体像

また，Cerena ら [3] はこれまでに生成されたトークンの調査を行い，トークンの窃取に関連するバックドア攻撃の 60%以上が，その寿命は 1 日よりも短いことを確認した．Huang ら [8] は，バックドア攻撃の調査を行い，7487 個のバックドアコントラクトを発見している．一般にスマートコントラクトのソースコードは使いまわしがされており [5, 18]，これらの研究では複製されたものの影響を受けている．それらの影響を取り除いた分析を行う点が，本研究の特徴である．

3. 問題設定

本節では本稿の問題設定を述べる．まず研究の全体像を述べたのち，具体的な問題設定としてスマートコントラクトの収集方法，及び，その静的解析について述べる．これらの設定の下，1 節で述べた問いを明らかにする．

3.1 研究手法の全体像

本稿では現実に悪性されたバックドア攻撃の影響を明らかにすべく，Ethereum ブロックチェーンに実際にデプロイされた全スマートコントラクトを収集し，そのソースコードを解析する．詳細は後述するとして，全体像を図 1 に示す．先ず，スマートコントラクトはその内容と実行が公開プラットフォームを通じて取得確認できることから，スマートコントラクトを既存のブロック探索用プラットフォームを通じて収集する．このとき，静的解析を行うこと，とくに後の工程でそのスマートコントラクトが作成された背景が類推しやすいよう，高級言語のソースコードが併せて公開されているものを対象とする．

次に，それらのスマートコントラクトのソースコードに対し，既存のバックドア検知ツール [13, 14] を適用することで，バックドアコントラクトの有無を検知する．このとき，各コンパイラバージョンの仕様がバックドア攻撃に影響していると考え，コンパイラバージョンごとに分析を試みる．試験的に分析を試みたところ，v0.5 以降のコンパイラで作成されたスマートコントラクトの分析は膨大な時間がかかることが分かった．一方，v0.4 までは比較的解析が容易に

© 2024 Information Processing Society of Japan

行えたこと, また, v0.4 は最近においてもスマートコントラクトが最も作成されたバージョンである [11] ことから, 本稿では v0.4 までのバージョンで調査を行う. v0.5 以降の調査は今後も継続して実施する.

3.2 静的解析

本稿ではスマートコントラクトのソースコードに着目した静的解析を行う. このとき, 新たなツールを設計することは目的とせず, 既存のツールを利用する. 具体的なツールとしては, 公開利用可能なものとして Pied-Pier [14] と CRPWarner [13] を用いる. 本稿では Solidity のソースコードがあるスマートコントラクトを対象としているが, いずれのツールもバイトコードでの解析が可能である. 本稿ではいずれかツールで検知された場合は, バックドアコントラクトとする. それぞれの大まかな機能を以下に述べる.

Pied-Pier [14] はファジングによる解析ツールである. スマートコントラクトのバイトコードからプログラムの中間表現を取得 [1] したのち, ファジング・モジュール [*3]を通じて, 攻撃に相当する状態に到達できるかを確認する. ArbitraryTransfer, GenerateToken, DestroyToken, DisableTransfer, FrozenAccount を検知できる.

CRPWarner [13] は情報フロー解析に基づく解析ツールである. まず EVM バイトコードを高級言語にデコンパイル [6] して制御フローグラフを取得したのち, 情報フロー解析を行った後, バックドア攻撃に直結する関数を特定する. ArbitraryTransfer, GenerateToken, DisableTransfer を検知できる[*4].

3.3 スマートコントラクトの収集

本稿では Google BigQuery に公開されている, Ethereum の mainnet におけるトランザクションのデータセット[*5]と, Ethereum ブロックチェーンの検索エンジンである Etherscan[*6]を利用する. 具体的には, BigQuery のデータセットにはコントラクトアドレスとそのアドレスへのトランザクションの情報が含まれている一方, 解析対象となるコードは含まれていないことから, BigQuery から取得したコントラクトアドレスを Etherscan に入力することで, スマートコントラクトを収集する.

本稿では Ethereum mainnet で一番最初のトランザクションが確認された 2015 年 8 月 7 日から本書執筆開始時点として 2024 年 5 月 31 日までに作成されたスマートコントラクトを収集した. コントラクトアドレスの重複を排除したところ, 66,283,849 件のスマートコントラクトが収集できた. スマートコントラクトでは一般にソースコードの使いまわしが多くされていることから [5, 18], これらのうちバイトコードが重複しているものを除外した. また, 一部のスマートコントラクトはコントラクト自体を自壊させる `self-destruct` 関数により, その中身が失われバイトコードが `0x` のみとなっていた. これらについても除外したところ, 1,271,127 件のスマートコントラクトが残った. 試験的に上述した静的解析のツールで評価したところ, 解析に膨大な時間を要することから, コンパイルバージョンを絞って解析することにした. 近年における利用が盛んなコンパイラバージョンとして v0.4 [11] までのバージョンを対象にしたところ, 83,239 件のスマートコントラクトが対象となった.

4. 調査結果

本節では調査結果を述べる. v0.4 までのコンパイラバージョンごとの違いについて, 表 1 に示す. なお, 表において全コントラクトの列は各コンパイラバージョンにおいて検知対象となったコントラクトの件数, ArbitraryTransfer から FrozenAccount の列における各数値は検知されたバックドアコントラクトの数, 同列内の括弧付きの数値は全コントラクト数の列に対する割合, 合計の列は何らかのバックドアコントラクトとして認定されたコントラクトの数を示す. 各列の総和と合計の列の数値が異なっている理由は, 複数のバックドアコントラクトとして検知されたコントラクトが存在するためである. それらのコントラクトは, いずれのバックドアコントラクトとしても計上している. 以降では, 複数のバックドアコントラクトとして検知されなかったもの, すなわち, 個々のバックドアコントラクトとしてのみ検知されたものを単一のバックドアコントラクトと呼称する. また, 複数種類のバックドアに相当するとして検知されたものは, 組み合わせバックドアコントラクトと呼称する.

本節の以降では, まず全体の傾向を述べたのち, 単一のバックドアコントラクトの傾向をそれぞれ述べる. 次に, 組み合わせバックドアコントラクトの傾向を述べる.

4.1 全体の傾向

表 1 によると, v0.1 がコントラクト数が少ないため割合が高くなっていることを除き, いずれのバックドアコントラクトにおいても, コンパイラバージョンが v0.2, v0.3, v0.4 と更新されるにつれて, バックドアコントラクトの数とその割合いずれも単調増加している. この単調増加の背景には, コンパイラバージョンが更新されるにつれて, ユーザ数が増加していることが考えられる. v0.1 から v0.3 は

[*3] `https://souffle-lang.github.io/index.html`

[*4] 文献 [13] では HiddenMintFunction, LeakingToken, LimitngSellOrder という名称になっているが, では HiddenMintFunction が GenerateToken, LeakingToken が ArbitraryTransfer, LimitngSellOrder が DisableTransfer に各要件がそれぞれ該当する.

[*5] `https://medium.com/google-cloud/full-relational-diagram-for-ethereum-public-data-on-google-bigquery-2825fdf0fb0b`

[*6] `https://etherscan.io/`

© 2024 Information Processing Society of Japan

表 1　コンパイラバージョンごとのバックドアコントラクトの数

	全コントラクト数	ArbitraryTransfer	GenerateToken	DestroyToken	DisableTransfer	FrozenAccount	合計
v0.1	28	4 (14.3%)	1 (3.6%)	0 (0%)	3 (10.7%)	0 (0%)	4 (14.3%)
v0.2	88	5 (5.7%)	4 (4.5%)	0 (0%)	5 (5.7%)	0 (0%)	9 (10.2%)
v0.3	556	78 (14.0%)	44 (7.9%)	0 (0%)	70 (12.6%)	0 (0%)	106 (19.1%)
v0.4	82567	24706 (29.9%)	21405 (25.9%)	717 (0.9%)	18789 (22.8%)	2560 (3.1%)	34863 (42.2%)
総数	83239	24793	21454	717	18867	2560	34983

スマートコントラクトの過渡期であり, v0.4 からが実用的なバージョンとして利用が増えている [11]. とくに v0.4 ではバックドアコントラクトの合計の数が 34983 件となり, 半数に近い数のコントラクトがバックドアコントラクトであることが確認された.

その内容としては, v0.4 では ArbitraryTransfer が最も多く, 24706 個のバックドアコントラクトが確認された. なお, DestroyToken と FrozenAccount はその数が他のバックドアコントラクトと比べて少ないように思われるが, これは Pied-Piper だけがこれらのバックドアコントラクトを検知する機能を持つことに起因する. すなわち, ArbitraryTransfer, GenerateToken, DisableTransfer は Pied-Piper と CRPWarner の両方で検知できるため, 母数が増えやすくなっている.

4.1.1　単一のバックドアコントラクトの傾向

前述した通り, 表 1 に含まれるコントラクトには, 複数のバックドアコントラクトとして検知されたものが多く存在している. 単一のバックドアコントラクトとして検知されたものの数を表 2 に示す. 表において全バックドアコントラクトの列は表 1 の合計の列に, 括弧付きの数値は全バックドアコントラクト数の列に対する割合にそれぞれ該当する. ArbitraryTransfer が 6530 個, GenerateToken が 4905 個, DisableTransfer が 28 個, DestroyToken が 100 個, FrozenAccount が 580 個である. 特徴的な点として, 単一のバックドアコントラクトとしての ArbitaryTransfer と GenerateToken は, 次節で述べる組み合わせバックドアコントラクトと比べてもその数が多い. これらは攻撃者が暗号資産を自分に転送する, あるいは, 自らが暗号資産を生成する点で, 単純に攻撃者が利益を得やすいことに起因すると考えられる.

一方, DistableTransfer はその母数に対して単一のバックドアコントラクトとしての数が 28 個しかないため, 割合の観点からも極めてその数が少ないといえる. このため, DisableTransfer は他のバックドアコントラクトと組み合わせることが前提と予想される. なお, DestroyToken と FrozenAccount も割合が少ないように思われるが, 上述した通りこれらのバックドアコントラクトは Pied-Piper だけがこれらのバックドアコントラクトを検知する機能を持つことに起因する.

4.1.2　組み合わせバックドアコントラクト

バックドアコントラクトのうち, 組み合わせバックドアコントラクトについて検知された種類ごとにグループに分けて可視化したものが図 2 である. 図の縦軸で ArbitraryTransfer を Arb, GenerateToken を Gen, DestroyToken を Des, DisableTransfer を Dis, FrozenAccount を Fro と表記しており, 各単語の連結はそれらをバックドアとして含んでいることを示している.

図によると, 組み合わせバックドアコントラクトはその種類に偏りがあり, DestroyToken と FrozenAccount を用いるものは数が少ない. これはこれらのバックドアコントラクト の母数が少ないことに起因している. 興味深いものは ArbitraryTransfer と DisableTransfer である. 前述した DestroyToken と FrozenAccount を含むものを除き, これらを用いたものは全てのすべての組み合わせバックドアコントラクトに含まれている. 前節でも述べた通り DisableTransfer は単一のバックドアコントラクトとして存在するものが 0.1% しかないため, やはり他のバックドアとの組み合わせが主な用途であることを示唆している.

また, すべてのバックドアコントラクトとして検知されたものが 56 個あった. 実際に発見したバックドアコントラクトの例として, ArbitraryTransfer, GenerateToken, DestroyToken, DisableTransfer, FrozenAccount 全てに検知されたものについて, コードの一部を簡潔に紹介する. 以下にそのコードの一部を掲載する.

```
1
2    function transferTo(address _to,uint
         amount) onlyOwner public returns(
         uint256 revenue) {
3        require(balanceOf[this] >= amount);
4        balanceOf[this] -= amount;
5        balanceOf[_to] += amount;
6        Transfer(this, msg.sender, amount);
7        revenue = balanceOf[this];
8        return revenue;
9    }
10
11   function burnFrom(address _from, uint256
         _value) public returns (bool success)
         {
12       require(balanceOf[_from] >= _value);
13       require(_value <= allowance[_from][msg
             .sender]);
```

© 2024 Information Processing Society of Japan

表 2　コンパイラバージョンごとの単一のバックドアコントラクトの数

	全バックドアコントラクト	ArbitraryTransfer	GenerateToken	DisableTransfer	DestroyTransfer	FrozenAccount
0.1	4	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
0.2	9	2 (22.2%)	2 (22.2%)	1 (11.1%)	0 (0%)	0 (0%)
0.3	106	8 (7.5%)	11 (10.4%)	0 (0%)	0 (0%)	0 (0%)(0%)
0.4	34863	6520 (18.7%)	4892 (14.0%)	27 (0.1%)	100 (0.3%)	580 (1.7%)
総数	34983	6530 (18.7%)	4905 (14.0%)	28 (0.1%)	100 (0.3%)	580 (1.7%)

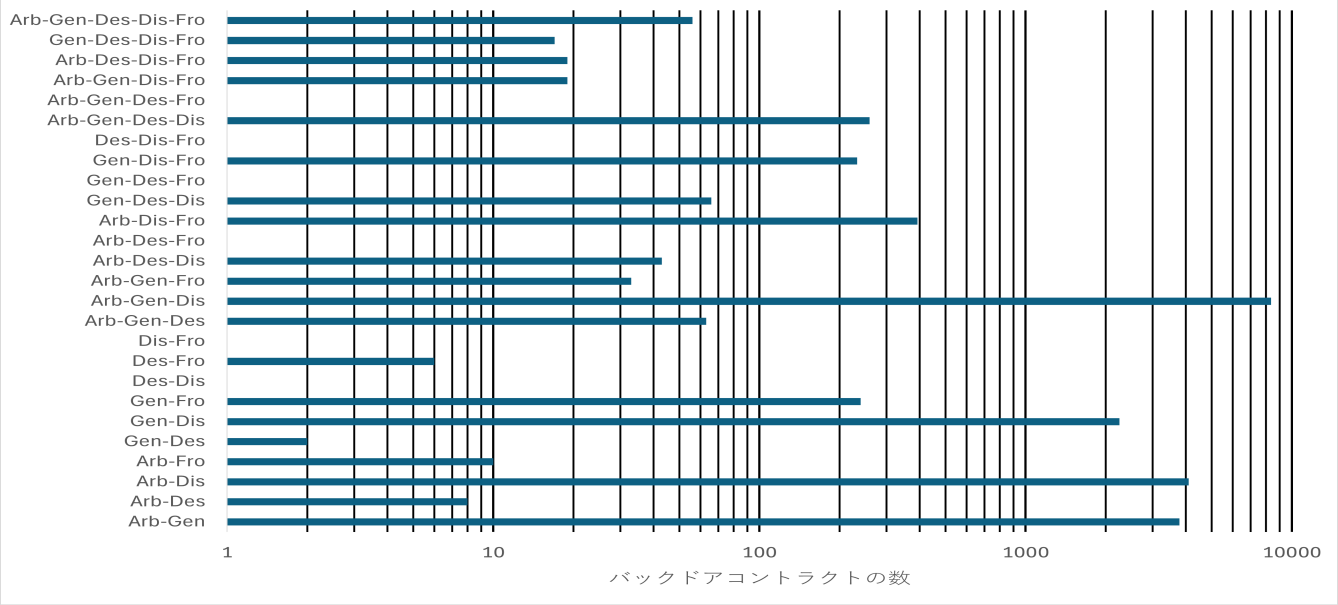

図 2　組み合わせバックドアコントラクトの数

```
14          balanceOf[_from] -= _value;
15          allowance[_from][msg.sender] -= _value
                ;
16          totalSupply -= _value;
17          Burn(_from, _value);
18          return true;
19      }
20
21      function freezeAccount(address target,
            bool freeze) onlyOwner public {
22          frozenAccount[target] = freeze;
23          FrozenFunds(target, freeze);
24      }
```

紙面上, 記載している関数は一部だけであるが, 上述した `transferTo` 関数が ArbitraryTransfer, `burnFrom` 関数が DestroyToken, `freezeAccount` 関数が FrozenAccount にそれぞれ該当する. これらの関数を含むためにバックドアコントラクトとして検知されたスマートコントラクトのなかには, スマートコントラクトにおけるトークン生成の挙動をデバックするために作られたと思しきものも見つかった. これらについては, スマートコントラクトの開発者が試験的に作ったところ, 偶発的にバックドアコントラクトと同様の挙動をするようになってしまったと考えられる.

5. 考察

本節では, これまで明らかにしたバックドア攻撃の存在について, 以下に述べる観点で詳細に分析する. まずは最も多かったバックドアコントラクトの内容, および, コードクローンによりバックドアコントラクトがどの程度作成されているか議論する. また, 本研究を実施するにあたって配慮した倫理的側面, 及び, 本稿における妥当性への懸念事項についても述べる.

5.1 最も多かったバックドアの種類

本節では, バックドアコントラクトの中でも数が多かったものについて述べる. 図 2 によると ArbitraryTransfer, GenerateToken, DisableTransfer を組み合わせたものが最も多く, 合計で 8,348 件発見した. 以下にその例として発見したバックドアコントラクトの一部を掲載する.

```
1   function add_referral(address referral,
        string promo, uint256 amount) external
         returns(address partner, uint256
        p_partner, uint256 p_referral){
2       p_partner = 0;
3       p_referral = 0;
4       partner = address(0x0);
5       if (msg.sender == contractPreICO ||
```

© 2024 Information Processing Society of Japan

```
            msg.sender == contractICO){
6           if(partnersPromo[promo] != address
                (0x0) && partnersPromo[promo]
                != referral){
7               partner = partnersPromo[promo
                    ];
8               referrals[referral] += amount;
9               amount_referral_invest +=
                    amount;
10              partnersInfo[partner].balance
                    += amount;
11              history[partner].push(
                    itemHistory(now, referral,
                     amount));
12              p_partner = (amount*uint256(
                    calc_partnerPercent(amount
                    )))/10000;
13              p_referral = (amount*
                    ref_percent)/10000;
14          }
15      }
16  }
```

この関数では, 5 行目の if 文内で msg.sender を指定することで関数の実行ユーザを実質的に制限するとともに, 8 行目で amount を増加させている. これらの挙動は文献 [14] で述べられている GenerateToken の挙動に酷似している. 紙面上詳細は割愛するが, ほかにも同様の挙動を含むバックドアコントラクトが確認されている. これらのバックドアコントラクトでは, 暗号資産の転送を制限すると同時に, 他のアドレスへ暗号資産を転送あるいは他のユーザに対しトークンを生成することで, 攻撃者が利益を得られるような挙動をすると考えられる.

5.2 コードクローンにおけるバックドアコントラクト

スマートコントラクトは一般に多くのコードが複製されている [5, 18] ことから, コードクローンの中にバックドアコントラクトが存在するか調査した. コンパイルバージョン v0.4 までの 83,239 件中のコントラクトにおいて, スマートコントラクトの機能を複製する ERC-1167[*7] により作成されたコードクローンに該当するものは 30 件確認できたが, いずれもバックドアコントラクトには該当しなかった.

その原因としては二つのことが考えられる. まずはトークンに関連するような既存のバックドア攻撃の寿命は 1 日以下と短いことである [3]. また, 本稿のコードクローンは ERC-1167 によるコードクローンを対象としており, ユーザが自ら手作業で開発するようなコードクローンあるいはコードは異なるが機能的に同一な意味的クローン [19] は含まれていない. これらを対象とした場合, コードクローンにバックドアコントラクトが含まれる可能性はある. これらの詳細な確認は, 今後の課題である.

[*7] https://eips.ethereum.org/EIPS/eip-1167

5.3 倫理的側面

本稿では過去にデプロイされたスマートコントラクトのソースコードを公開プラットフォームで収集したのみであり, 新たにバックドアコントラクトの作成や非公開の情報を公にするといったことはしていない. このため, 新たに被害者を誘引することはない. また, 解析においてもスマートコントラクトのソースコードを静的解析したのみであり, 既存のバックドアコントラクトにトランザクションを送信するといったことはしていない. このため, 攻撃者が金銭的利益を得るような活動はしていない. 本稿の調査は特定のスマートコントラクトの脆弱性を指摘するなど, 新しい攻撃を示したわけではない. そのため, 第三者による攻撃を助長することも, 新たに不利益を講じるユーザが発生するようなこともしていない. 以上のことから, 本稿の調査は新たな攻撃を助長することも, 被害を増やすこともしていないといえる.

5.4 妥当性への懸念事項

本稿における懸念事項は大きく 3 点である. それぞれ以下に述べる.

まず, スマートコントラクトのコンパイラバージョン v0.5 以降については, 本稿では解析時間の制約上, 含めることができなかった. 本稿では v0.4 が最も利用されているバージョンの一つ [11] として焦点を当てたが, 最新のバージョンである v0.8 では異なる傾向が得られる可能性もある. v0.5 から v0.8 で作成された残りの 1,187,888 件に関する分析は今後の課題である.

調査で用いたツール内でエラーが発生し解析できないスマートコントラクトが 1800 件ほどあった. これらのエラー原因はその構造の複雑さに起因すること, また, 一般には洗練されたバックドアの設計であれば構造が複雑となることが知られている [14]. すなわち, これらの中にもバックドアコントラクトが存在する可能性はあることから, 今後はこれらのエラー対応したうえで, より詳細な分析を行う.

最後に, 本稿の分析結果は CRPWarner と Pied-Piper に依存している. すなわち, これらのツールで検知漏れあるいは過検知となっているバックドアコントラクトが存在する可能性はある. 文献 [17] によると少なくとも検知漏れについては他のツールを追加導入することで検知漏れは防げるようになることから, 今後は更なる分析ツールを導入した分析を予定している.

6. まとめ

本稿では Ethereum スマートコントラクトにおいてバックドア攻撃がどれだけ行われているか, Ethereum の全スマートコントラクト 66,283,849 件を収集し, そのソース

© 2024 Information Processing Society of Japan

コードを解析することで実態調査した. 最も利用されているコンパイラバージョンである v0.4 までのバージョンにおいて, 合計 34,983 件のバックドアコントラクトを発見した. これは当該バージョンにおける 42.2% のスマートコントラクトがバックドアであることを意味している. 最も作成されていたバックドアは ArbitraryTransfer であり, 24,793 件のバックドアコントラクトを発見した. バックドアコントラクトには他の手法との組み合わせを前提としているものもあり, DisableTransfer は 99.9%以上が他のバックドアコントラクトとの組み合わせだった. 最も多かったバックドアコントラクトの種類は, 上記の ArbitraryTransfer と DisableTransfer に加え, さらに GenerateToken を組み合わせたものであり, 8,348 件発見している. これらの理由については, 攻撃者が転送を任意に制御することで, 自らの資産を増やす手法が多いことを示唆している. 今後の課題は v0.5 から v0.8 のコンパイラバージョンにおいても分析を行うこと, また, 他のツール [9, 15, 16, 25] も用いたより精密な分析が挙げられる.

謝辞 本研究の一部は JST さきがけ事業 (課題番号: JPMJPR23P6) により支援されている.

参考文献

[1] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz. Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*, 2018.

[2] M. Cao, Y. Zhang, Z. Feng, J. Hu, and Y. Zhu. Tokenauditor: Detecting manipulation risk in token smart contract by fuzzing. In *Proc. of QRS 2022*, pages 651–662. IEEE, 2022.

[3] F. Cernera, M. La Morgia, A. Mei, and F. Sassi. Token spammers, rug pulls, and sniper bots: An analysis of the ecosystem of tokens in ethereum and in the binance smart chain ({{{{{BNB}}}}}). In *PRoc. of USENIX Security 2023*, pages 3349–3366. Usenix Security, 2023.

[4] H. Chen, M. Pendleton, L. Njilla, and S. Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys*, 53(3):1–43, 2020.

[5] X. Chen, P. Liao, Y. Zhang, Y. Huang, and Z. Zheng. Understanding code reuse in smart contracts. In *Proc. of SANER 2021*, pages 470–479. IEEE, 2021.

[6] N. Grech, L. Brent, B. Scholz, and Y. Smaragdakis. Gigahorse: thorough, declarative decompilation of smart contracts. In *Proc. of ICSE*, pages 1176–1186. IEEE, 2019.

[7] K. e. a. Hara. Machine-learning approach using solidity bytecode for smart-contract honeypot detection in the ethereum. In *QRS-C*, pages 652–659. IEEE, 2021.

[8] J. Huang, N. He, K. Ma, J. Xiao, and H. Wang. A deep dive into NFT rug pulls. *CoRR*, abs/2305.06108, 2023.

[9] R. Ji, W. Wang, Y. Xiong, and W. Huang. Solscope: Effectively hunting potential permission backdoor threats in smart contracts. In *2023 9th International Conference on Big Data Computing and Communications (BigCom)*, pages 88–95. IEEE, 2023.

[10] C. Kado, N. Yanai, J. P. Cruz, K. Yamashita, and S. Okamura. An empirical study of impact of solidity compiler updates on vulnerabilities in ethereum smart contracts. *CoRR*, abs/2306.04250, 2023.

[11] C. Kado, N. Yanai, J. P. Cruz, K. Yamashita, and S. Okamura. An empirical study of impact of solidity compiler updates on vulnerabilities in ethereum smart contracts. *CoRR*, abs/2306.04250, 2023.

[12] S. Kaur, S. Singh, S. Gupta, and S. Wats. Risk analysis in decentralized finance (defi): a fuzzy-ahp approach. *Risk Management*, 25(2), 2023.

[13] Z. Lin, J. Chen, J. Wu, W. Zhang, Y. Wang, and Z. Zheng. Crpwarner: Warning the risk of contract-related rug pull in defi smart contracts. *IEEE Transactions on Software Engineering*, 50(6):1534–1547, 2024.

[14] F. Ma, M. Ren, L. Ouyang, Y. Chen, J. Zhu, T. Chen, Y. Zheng, X. Dai, Y. Jiang, and J. Sun. Pied-piper: Revealing the backdoor threats in ethereum erc token contracts. *ACM Transactions on Software Engineering and Methodology*, 32(3):1–24, 2023.

[15] B. Mazorra, V. Adan, and V. Daza. Do not rug on me: Leveraging machine learning techniques for automated scam detection. *Mathematics*, 10(6):949, 2022.

[16] M. H. Nguyen, P. D. Huynh, S. H. Dau, and X. Li. Rug-pull malicious token detection on blockchain using supervised learning with feature engineering. In *Proc. of ACSW 2023*, pages 72–81. ACM, 2023.

[17] D. Perez and B. Livshits. Smart contract vulnerabilities: Vulnerable does not imply exploited. In *Proc. of USENIX Security 2021*, pages 1325–1341. USENIX Association, 2021.

[18] G. A. Pierro and R. Tonelli. Analysis of source code duplication in ethreum smart contracts. In *Proc. of SANER 2021*, pages 701–707. IEEE, 2021.

[19] C. K. Roy and J. R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEEN' S UNIVERSITY*, 115, 2007.

[20] T. Sharma, R. Agarwal, and S. K. Shukla. Understanding rug pulls: an in-depth behavioral analysis of fraudulent nft creators. *ACM Transactions on the Web*, 18(1):1–39, 2023.

[21] D. Sun, W. Ma, L. Nie, and Y. Liu. Sok: Comprehensive analysis of rug pull causes, datasets, and detection tools in defi. *CoRR*, abs/2403.16082, 2024.

[22] G. Wood. Ethereum: A secure decentralised generalised transaction ledger byzantium version. `https://ethereum.github.io/yellowpaper/paper.pdf`, 2022.

[23] P. Xia, H. Wang, B. Gao, W. Su, Z. Yu, X. Luo, C. Zhang, X. Xiao, and G. Xu. Trade or trick? detecting and characterizing scam tokens on uniswap decentralized exchange. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(3):1–26, 2021.

[24] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais. Sok: Decentralized finance (defi) attacks. In *Proc. of IEEE S&P 2023*, pages 2444–2461. IEEE, 2023.

[25] Y. Zhou, J. Sun, F. Ma, Y. Chen, Z. Yan, and Y. Jiang. Stop pulling my rug: Exposing rug pull risks in crypto token to investors. In *Proc. of ICSE-SEIP 2024*, pages 228–239. ACM, 2024.

© 2024 Information Processing Society of Japan