# イーサリアムコントラクトハニーポットのリスク分析

石巻 東哉[1,a)]　面 和成[1,2]

**概要**：近年，Ethereum プラットフォームを活用したスマートコントラクトが注目を集めているが，暗号資産の窃盗を目的としてスマートコントラクトがサイバー攻撃の標的となるケースが増加している．攻撃手法の一つとして，一見脆弱性を含むように思われるが，バッグドアが存在するコントラクトを意図的に展開することにより，脆弱なコントラクトを狙っている他の攻撃者を誘い込み，暗号資産を盗み取るという手法が登場している．これはスマートコントラクトハニーポットと呼ばれる（以降，単にコントラクトハニーポットと呼ぶ）．Torres らは USNIX Security 2019 において初めてコントラクトハニーポットを分析し，その検知手法について報告した．本研究では，Torres らが整理した 8 種類のコントラクトハニーポットを対象とし，それぞれについての被害状況を明らかにすると共に，コントラクトハニーポットの送金処理の引数に着目した分析を行い，新たなコントラクトハニーポットを発見した．

**キーワード**：ブロックチェーン，Ethereum，スマートコントラクト，ハニーポット， リスク分析

# Ethereum Contract Honeypot Risk Analysis

Motoya Ishimaki[1,a)]　Kazumasa Omote[1,2]

***Abstract:*** In recent years, smart contracts utilizing the Ethereum platform have attracted much attention, and smart contracts are increasingly becoming targets of cyber attacks for the purpose of stealing cryptographic assets. One of the emerging attack methods is to intentionally deploy contracts that appear to contain vulnerabilities but have bagdoors, thereby luring other attackers who are targeting vulnerable contracts to steal cryptographic assets. These are called smart contract honeypots(henceforth referred to simply as contract honeypots); Torres et al. analyzed contract honeypots for the first time at USNIX Security 2019 and reported on their detection methods. In this study, we target eight types of contract honeypots organized by Torres et al. and clarify the damage status of each of them. We also analyze contract honeypots focusing on the arguments of the money transfer process of contract honeypots and discover a new type of contract honeypot.

***Keywords:*** Blockchain, Ethereum, Smart Contract, honeypot, Risk Analysis

## 1. はじめに

近年, 多くの資産を分散化可能な技術の一つとしてブロックチェーンが注目されている．Bitcoin [1] と Ethereum [2] が最も有名な 2 つある．この Ethereum を活用し，暗号資産以外にもブロックチェーン技術を用いた様々なサービスが登場している．これらサービスにとって重要になるのがスマートコントラクトである．スマートコントラクトとは第三者機関の介入なしで自動的に契約を執行することが可能なシステムのことである．また，非中央集権であり，改ざん不能なサービスを実現することができる．

しかしながら，暗号資産は匿名性があり，サイバー攻撃の標的となるケースが増加している．Ethereum ブロックチェーンの特徴であるスマートコントラクトは通常，Solidity を代表とするプログラミング言語を使用して実装

[1] 筑波大学
University of Tsukuba
[2] 情報通信研究機構
National Institute of Information and Communications Technology, Japan
[a)] s2120527@s.tsukuba.ac.jp

される．これにより，他の一般的なソフトウェアと同様にバグによる脆弱性が含まれる可能性がある．いくつかのスマートコントラクトは過去に既に攻撃されており，The DAO 事件では約 60 億円の損失が生じている．また，スマートコントラクトはブロックチェーンを活用しているため，一度ブロックチェーン上に展開され格納されたコード，そしてその実行結果を改変することは非常に困難である．これらのことから，攻撃者は脆弱なスマートコントラクトを探し始め，暗号資産の盗難を狙うようになっており，それを対策するべく様々なアプローチで研究がされている [4] [5] [6].

しかしながら，上記の手口に留まることなく，攻撃者は更に巧妙な手口を探っている．Torres らの研究 [7] では，攻撃者が脆弱なコントラクトを探すのではなく，あえて設置側になることで，自分より脆弱な攻撃者から資産を騙し取ろうとしていることが明らかになっている．これはスマートコントラクトハニーポットと呼ばれる．（以降，単にコントラクトハニーポットと呼ぶ．）このコントラクトハニーポットを検出すべく，Torres ら [7] はコントラクトハニーポットの検知を行うことが可能な HoneyYBadger と呼ばれるツールを提案した．

本研究では，Torres ら [7] が整理した 8 種類のコントラクトハニーポットを対象とし，それぞれについて被害件数及び被害額の推移を明らかにするとともに，コントラクトハニーポットの新たな特徴を分析し，新種のコントラクトハニーポットを発見した．このとき，送金処理（msg.sender.transfer）の引数に着目し，「this.balance」という引数がコントラクトハニーポットの特徴となっていることを突き止めた．さらに，本研究は先行研究では行われていなかった一般ユーザーへのリスク分析を行った．

## 2. 準備

### 2.1 スマートコントラクト

スマートコントラクトは，第三者機関の介入なしで自動的に契約を執行することが可能なシステムのことを指す．これを Ethereum プラットフォーム上で実現することで，サービスの提供者と利用者間の交換契約内容をブロックチェーン上にプログラミングし，契約条件が満たされると自動的に取引が行われるシステムを構築することが可能である．スマートコントラクトは，以下のようなメリットがある．

- 取引内容の不正や改ざんが行われにくい．
- 取引内容の透明性が高い．
- 非中央集権的にサービスを運用することが可能であり，システムダウンのリスクを抑えることができる．

このように多くのメリットがあるが，契約時に関与する暗号資産を管理する必要があるため，これにより攻撃者の攻撃モチベーションを高め，標的の原因となっているのが現実である．

### 2.2 Solidity

Solidity は，コントラクトを記述することに特化した高水準言語である．ユーザーはコントラクトをコンパイルして，トランザクションにバイトコードの形で格納し，スマートコントラクトとしてデプロイする．また，格納後にコンパイル前の Solidity ソースコードを確認することは本来できない．しかし，スマートコントラクトの作成者は自らのコントラクトを周知して利用してもらうことを目的としているため，EtherScan と呼ばれる Ethereum ブロックチェーンの状態を GUI で確認することが可能な Web サイトにソースコードを登録している場合がある．コントラクトハニーポットの作成者も他の攻撃者に脆弱性の存在を発見してもらう必要があるため，登録を行う傾向がある．

### 2.3 コントラクトハニーポット

あえて設置側になることで自分より脆弱な攻撃者から資産を騙し取ろうとしている攻撃者が登場している．彼らは誰でも悪用可能な Solidity コードをコントラクトに含ませ，そのコントラクトにトラップドアを隠した状態でコントラクトを意図的に展開することにより，悪用できるコントラクトを狙っている他の攻撃者を誘い込み，おびき寄せられた攻撃者から暗号資産を騙し取るという手法をとる．このように，コントラクトハニーポットは，一見容易に悪用できそうではあるが実際は悪用できないように巧妙に作られているトラップドアを持つ Solidity コードである．表 1 にコントラクトハニーポット詳細を示す．Torres らの研究 [7] によるとコントラクトハニーポットは大きく 8 種類に分けられ，コントラクトハニーポット毎に脆弱性を悪用する技術が異なる．また，その技術は大きく分けて 3 種類に分けられることが明らかになっている．

## 3. 関連研究

### 3.1 シンボリック実行を用いた検出

Torres ら [7] は Solidity コードをベースとしてコントラクトハニーポットの検知を行うことが可能な HoneyBadger[*1] と呼ばれるツールを発表している．筆者らは目視でスマートコントラクトを確認することで，各コントラクトがコントラクトハニーポットか否かのラベル情報・8 種類のハニーポットのトラップドア手法の情報を公開している．この詳細は表 1 となる．しかし, 筆者らの研究は検出時間の遅さや事前定義したパターンが必要というデメリットをもっている．さらには一般ユーザーに対するコントラクトハニーポットのリスク分析までには至っていないことがあ

[*1] https://github.com/christoftorres/HoneyBadger

表 1　コントラクトハニーポット

| トラップドア名 | 対象技術 | 内容 |
|---|---|---|
| Balance Disoder (BD) | EVM | コントラクトの実行前に残高が変動することを利用 |
| Inheritance Disoder (ID) | Solidity Compiler | コントラクトが他のコントラクトを継承する際の仕様を利用 |
| Skip Empty String Literal (SESL) | | 空の文字リテラルがスキップされるという言語特性を利用 |
| Type Deduction Overflow (TDO) | | コンパイラの型推論機能を利用 |
| Uninitialised Struct (US) | | 構造体の仕組みを利用 |
| Hidden State Update (HSU) | EtherScan Blockchain Explorer | 空のトランザクション値を含む内部メッセージは表示されない仕様を利用 |
| Hidden Transfer (HT) | | コード長が一定数を超えると表示されないという仕様を利用 |
| Straw Man Contract (SMC) | | リエントラント攻撃に対して脆弱なように見せかける |

げられる．

### 3.2　バイトコードを用いたコントラクトハニーポット検出

Chen ら [8] はスマートコントラクトハニーポットを検出するために EVM バイトコードを用いて，スマートコントラクトの検知を行った．彼らが検知に用いた特徴は CALLER や BALANCE という特徴量である．しかし，これらの特徴はアカウントの残高確認に関する特徴，アカウントのアドレス確認に関する特徴などのスマートコントラクトを実行しないと分からない特徴を用いていることから，暗号資産の盗難発生前に検知できないため，未然に被害を防ぐのは困難であると考えられる．また，この研究は一般ユーザーに対するリスク分析までには至っていない．

### 3.3　機械学習を用いたコントラクトハニーポットの検出

Camino ら [3] は，シンボリック実行による検知時間の遅さのデメリットを解消するためにスマートコントラクトに紐づく取引履歴をはじめとする情報を特徴量として，機械学習モデル（XGBoost）を学習することで検知を行うというアプローチを提案した．彼らが使用した特徴量は Torres ら [7] と同じ特徴量である．Camino らの研究において使用されている特徴量である Transaction，Fund Flow は十分にコントラクトが動作した後に入手可能な情報であるため，実際に被害を抑えるためには活用することができない．一方で，Hara ら [10] は Camino らの課題を踏まえ，コントラクトの動作前に検知すべくバイトコードの特徴を用いてコントラクト実行前の検知を検討するモデルを構築したが，XGBoost のみの検知のため他の機械学習アルゴリズムとの比較をする必要がある．また，Teng ら [9] はトランザクションをベースとしたコントラクトハニーポットの検知を検討したが，これは自己破壊したコントラクトハニーポットも検知する可能性が示唆されている．自己破壊されたコントラクトはトランザクションとしては機能しないため，リアルタイムな検知ができない．これらの研究は全て一般ユーザーに対するリスクの分析までには至っていない．

## 4.　コントラクトハニーポット

### 4.1　概要

Torres らの研究では，コントラクトハニーポットは 8 種類に分類されている．本節では，特にコントラクトハニーポットのコード特性に基づいて 8 つのコントラクトハニーポットについて説明する．

### 4.2　Balance Disorder (BD)

ユーザーはコントラクトハニーポットに含まれている残高以上のイーサを送金して残高と送金分全てを搾取しようと試みている．しかし，このコントラクトハニーポットでは，送金分がコントラクト実行前に残高に増分されてしまうため，送金条件が一致せずに送金分が窃取されてしまう．

### 4.3　Inheritance Disorder (ID)

Solidity における継承を悪用したコントラクトで, 窃取にはコントラクト所有者を変更する必要がある．所有者を変更できるように思わせる Solidity コードを意図的に含ませるが，変数のスコープを悪用しているため，コントラクト設置者を変更する処理を参照することができず，コントラクト所有者を変更することができずにイーサが搾取される．

### 4.4　Skip Empty String Literal (SESL)

空文字列をあえて関数の引数に埋め込むことで Solidity コンパイラのエンコード処理をスキップさせ，後ろに続く関数の引数を 32 バイト分ずらすコントラクトハニーポットである．この機能により, 送金分のイーサは返ってこず, イーサがコントラクトハニーポット設置者に盗み取られるという仕組みである．

### 4.5　Type Deduction Overflow (TDO)

変数の型を悪用したコントラクトハニーポットであり，ループの停止条件に達することができた場合にイーサを手に入れられるという仕組みを悪用したものである．しかし，手に入る最大のイーサは 255 wei であり，送金した分よりも必然的にかなり低い値が返ってくる．

### 4.6 Uninitialised Struct (US)

このコントラクトハニーポットは，ユーザーが残高を引き出すために，最小限をベットし，コントラクトに格納されている乱数を推測する必要がある．しかし，ブロックチェーンに保存されているデータはすべて公開されている．どのユーザーも簡単に乱数の値を取得することができるため，乱数を被害者に推測させる guessNumber を呼び出して掛け金をベットするように誘発する．しかし，一般的にはコントラクトハニーポットの構造体は初期化されていないため，構造体に含まれる最初の変数の格納場所が，乱数を表す randomNumber の格納場所にマップされることにより，乱数が呼び出し元のアドレスにより上書きされ，送金条件を満たしにくくする．

### 4.7 Hidden State Update (HSU)

見かけ上，イーサを引き出すための変数が False か True になっていることを被害者に思い込ませ，攻撃者が暗黙的に関数を実行することによりその変数を上書きする．変数の変更に気づかない被害者が知らないうちにイーサを搾取される．

### 4.8 Hidden Transfer (HT)

以前の Etherscan は Solidity コードを表示するスクロールボックスに水平スクロールと垂直スクロールの機能を持ち合わせおり，EtherScan のスクロール機能を悪用したコントラクトハニーポットである．ソースコードの後ろに非常に長いスペースを挿入することによって，Etherscan 上で表示されるコードを効果的に隠蔽することができる．この手法によりトラップを隠すことで被害者を欺くという趣旨のコントラクトハニーポットである．

### 4.9 Straw Man Contract (SMC)

リエントリ攻撃の脆弱性を悪用したコントラクトハニーポットである．”CashOut”という文字列を送金処理に含ませることにより，呼び出し元をコントラクトハニーポット設置者に限定し，それ以外のユーザーは例外が発生するため，攻撃が失敗し，コントラクトに送金した分のイーサが窃取される．

## 5. 分析

### 5.1 分析の概要

本研究では，先行研究ではなされていないコントラクトハニーポットの被害状況の分析をし，ブロックチェーンユーザーにコントラクトハニーポットの危険性を示唆することを目的とする．また，コントラクトハニーポットに用いられる Solidity のクラス名やイーサを送金する関数 msg.sender.transfer の引数を抽出することで攻撃者の意図を把握する．

### 5.2 分析に用いたデータ

本研究で扱うコントラクトハニーポットは, HoneyBadger [7] で検知された 2017 年〜2019 年分のコントラクトハニーポット計 323 件とする．一方，HoneyBadger で検知されなかったコントラクトを正規コントラクトと見なし，最も被害が多かった 2018 年 1〜2 月分のコントラクト計 263,007 件を収集し，そのうちソースコードに msg.sender.transfer を持つ 771 件を本分析で扱う正規コントラクトとする．

### 5.3 コントラクトハニーポットの被害状況

#### 5.3.1 分析目的

Torres ら [7] の研究では，8 種類のコントラクトハニーポットの分析及び検知方法の提案を行っているが，それぞれのコントラクトハニーポットが実際に被害にあったのか，また被害になっているのならばどの程度の被害額であったのかは明らかになっていない．そこで本分析では，HoneyBadger [7] でコントラクトハニーポットと判定されたコントラクトを対象に，実被害件数及び実被害額を明らかにする．

#### 5.3.2 分析方法

コントラクトハニーポット被害状況の分析方法としては各月のコントラクトの被害件数・被害額を EtherScanAPI[*2] により収集し，時系列で各月の被害件数・被害額の推移を算出する．

#### 5.3.3 分析結果

コントラクトハニーポットの総被害の内訳を表 2 に示す．また，コントラクトハニーポットの被害件数と被害額の推移を時系列分析したものを図 1，図 2 に示し，1 件当たりの被害額の平均を分析したものを図 3 に示す．表 2 によると，HSU は総被害額が約 60Ether と全てのコントラクトハニーポットの中で最も高く，被害件数も 74 件と最も大きい．TDO は被害件数が 1 件で被害額が 1.00 E-07 と他のコントラクトハニーポットと比べて被害が少なかった．図 3 によると，ID は，1 件当たりの被害額が最大で約 7ETH と他のコントラクトハニーポットとは大きく異なっていた．さらに，特定の期間に着目したところ，2018/03 以降全体的にはコントラクトハニーポットの被害数もまばらにはなったが，HSU の被害件数と被害額は 2018/03〜2018/11 にかけて激しい増減を示した（図 1，図 2）．

### 5.4 コントラクトハニーポットの特徴分析

#### 5.4.1 分析目的

Torres ら [7] の研究では，機械学習を用いてコントラクトハニーポットのコードを分析して特徴を抽出している．しかし，コードを分析せずともコントラクトハニーポットの特徴が現れやすいものにコントラクト名と

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

表 2　コントラクトハニーポットの総被害の内訳

| トラップドア名 | 被害件数 | 被害総額 (Ether) |
|---|---|---|
| Balance Disoder (BD) | 10 | 4.5101 |
| Inheritance Disorder(ID) | 27 | 18.22660389 |
| Skip Empty String Literal(SESL) | 12 | 6.599685965 |
| Type Deduction Overflow (TDO) | 1 | 1.00E-07 |
| Uninitialised Struct (US) | 35 | 7.036012884 |
| Hidden State Update (HSU) | 74 | 59.93716123 |
| Hidden Transfer(HT) | 2 | 1.1 |
| Straw Man Contract (SMC) | 16 | 7.664034233 |

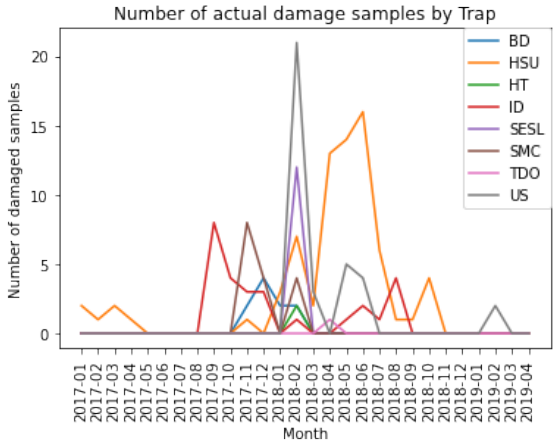


**図 1**　各コントラクトハニーポットの被害件数の推移

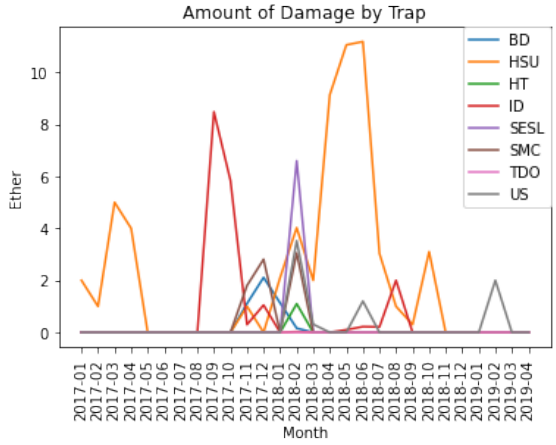


**図 2**　各コントラクトハニーポットによる被害額の推移

送金処理（msg.sender.transfer）の引数名が考えられる．msg.sender.transfer とはイーサを送金するための関数であるため，その関数の引数が注目される．そこで本分析では，使用頻度の多いコントラクト名及び msg.sender.transfer の引数名を明らかにする．

### 5.4.2　分析手法

8 つのコントラクトハニーポットを対象に EtherScanAPI により収集し，全てのコントラクトに含まれるコントラクト名と msg.sender.transfer に含まれる引数の頻度を算出する．

### 5.4.3　分析結果

図 4 はコントラクトに含まれるコントラクト名の頻度を

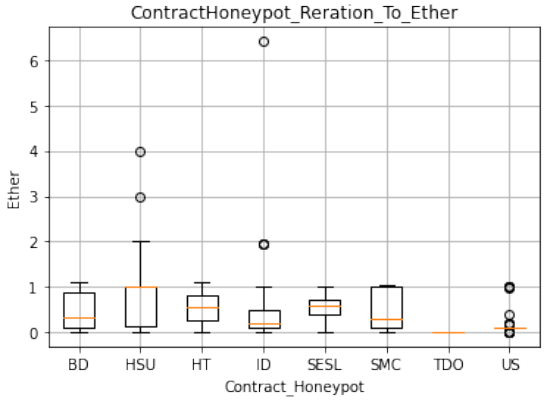


**図 3**　各コントラクトハニーポットの 1 件当たりの被害額

算出したものであり，図 5 は全てのコントラクトに含まれる msg.sender.transfer の引数の頻度を算出したものである．図 4 から，BD，HSU，ID，SESL，US の 5 種類のコントラクトハニーポットに共通して ownalbe をコントラクト名に用いていることが分かった．また図 5 から，8 種類中 7 種類のコントラクトハニーポットの msg.sender.transfer の引数に this.balance が頻出していることが分かった．

ほとんどのコントラクトハニーポットに共通していることとして，振り込まれている全てのイーサ残高を連想する this.balance を msg.sender.transfer の引数に用いる傾向にあり，さらにイーサを取得できることを表す ownable といったコントラクト名を多用していた．

## 5.5　新たなコントラクトハニーポットの検知

### 5.5.1　分析目的

前節において，コントラクトハニーポットの送金処理（msg.sender.transfer）の引数として this.balance が頻出していることが明らかになった．そこで，この引数を用いて，HoneyBadger が検知できなかったコントラクトハニーポットを検知できるかを試みる．

### 5.5.2　分析手順

分析手順を次に示す．

(1) 2018 年 1 月〜2 月分のコントラクトの収集

(2) msg.sender.transfer の引数に this.balance を伴うコントラクトの収集

表 3　HoneyBadger で未検知のコントラクトハニーポット

| コントラクトアドレス（上位 5 桁） | デプロイされた日時 | 被害額 (Ether) |
|---|---|---|
| 0x31FD6... | 2018/1/31 12:53 | 1.01 |
| 0x5ABb8... | 2018/1/31 15:59 | - |
| 0x2BB5B... | 2018/1/31 16:26 | - |
| 0x55Bec... | 2018/2/1 3:52 | - |
| 0x61dc3... | 2018/2/1 4:55 | - |
| 0x66385... | 2018/2/1 19:29 | 1.01 |
| 0xf414b... | 2018/2/2 10:01 | - |
| 0xcfebf... | 2018/2/2 10:28 | 1.05 |
| 0x0b82b... | 2018/2/6 7:35 | - |

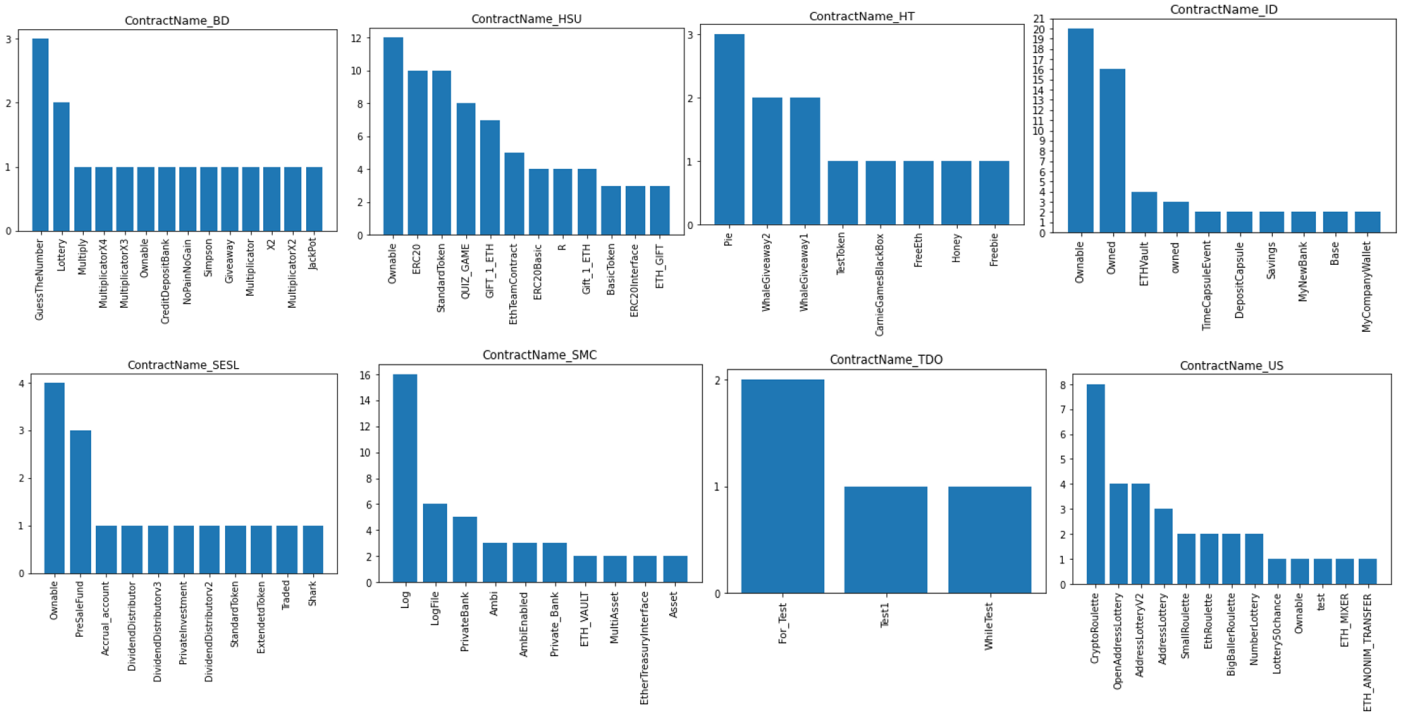


**図 4**　各コントラクトにおけるコントラクト名

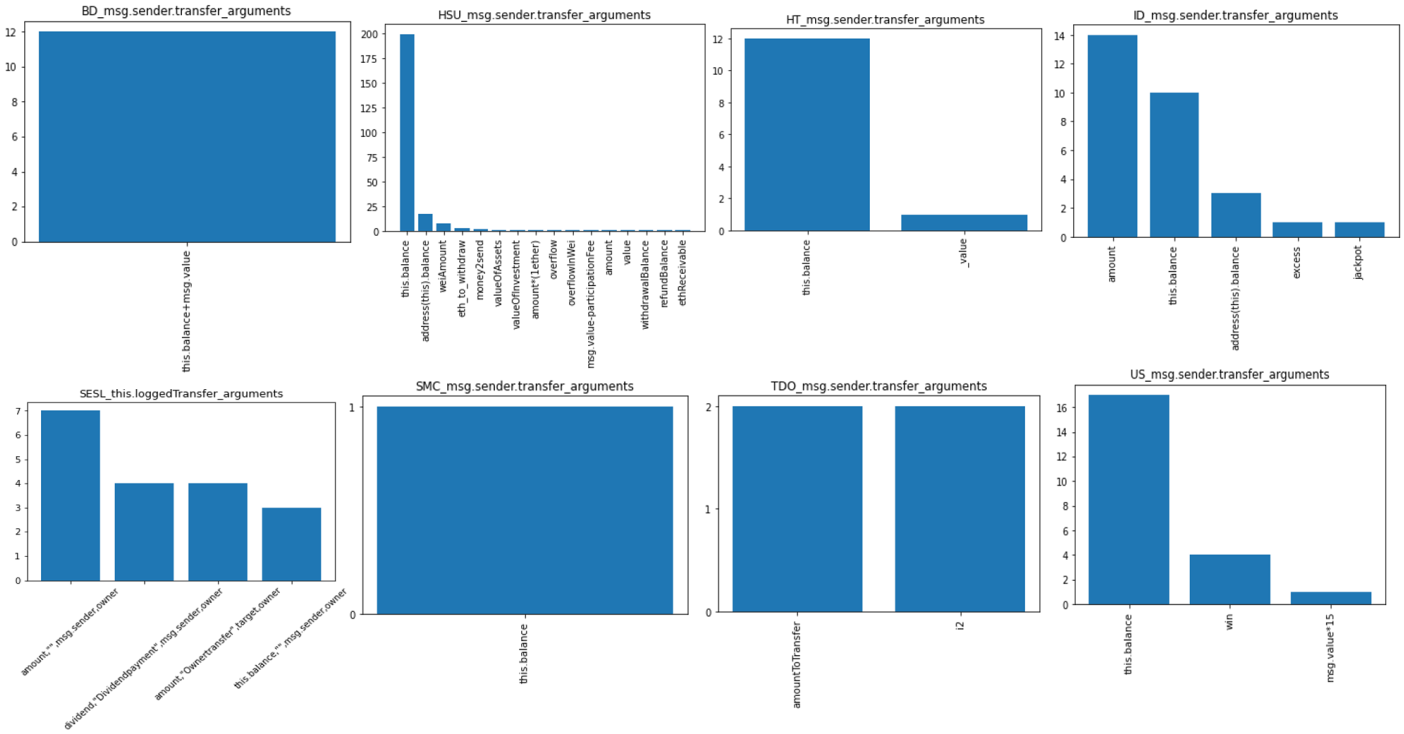


**図 5**　各コントラクトにおける引数名

### 5.5.3　分析結果

本分析では，msg.sender.transfer の引数として this.balance を含む正規コントラクトを 50 件を発見し，うち 3 件は同一の Solidity ファイルで記述されていたため，47 件を分析対象とした．そして，this.balance による指標で HoneyBadger では検知できなかったコントラクトハニーポットを 47 件のうち 9 件を発見した（表 3）．なお，この 9 件のうち 3 件と 2 件のコントラクトの内容は全て同じであったため，コピーされて用いられていたと考えられる．また，HT と思われるコントラクトハニーポット 3 件と HT と HSU の機能を組み合わせたコントラクトハニーポット 3 件を確認した．さらに，送金処理に if(1==2) を

伴うコントラクトハニーポット 3 件を発見した．これら 3 件は Torres ら [7] による分類手法にはどれにも当てはまらなかったため新しいコントラクトハニーポットだと考えられる．また，9 件中 3 件には計 3.07Ether の被害が 2018/1/31 13:57，2018/2/2 7:15，2018/2/6 10:14 に確認できた．

### 5.6 正規コントラクトについて

#### 5.6.1 分析目的

正規コントラクトにおいて，this.balance がどの程度出現するのか，またその正規コントラクトはどのような内容であるのかを明らかにする．

#### 5.6.2 分析手順

分析手順はゲーム型のコントラクトかトラップ型のコントラクトかなどの Solidity コードの種類を確認する．

#### 5.6.3 分析結果

図 6 に正規コントラクトにおける msg.sender.transfer の引数調査結果を示す．正規コントラクトにおいては balance という引数が最も多く使用されていることが分かった．この引数は大きく分けて，ゲーム型コントラクトにおける送金者の掛け金の総額に関する引数やコントラクト設置者がゲーム参加者に支払う金額の引数の 2 種類に分けられ，コントラクトの全残高を表す this.balance とは意味合いが異なる．また，amount のような残高を連想させる引数名の使用も目立ったが，そのうちのほとんどが送金者が支払う金額やイーサを引き出す金額に関する変数であることが分かり，コントラクトハニーポットを示す指標にはなり得ないと考える．

msg.sender.transfer に this.balance を含む正規だと思われるコントラクトは 47 件中 38 件であった．そのうち 23 件が目的が分からないコントラクトや犯罪に関わり得るようなコントラクトの存在が確認できた．具体的には，賭博や数あてなどのゲーム型コントラクトが 14 件，マルチ商法型のコントラクトが 1 件，スマートコントラクトの所有権の売買に関するコントラクトが 3 件，送金型のコントラクトが 5 件であった．また，残り 15 件については，両替型のコントラクトが 5 件，クラウドファンディング型コントラクトが 1 件，トークン売買に関するコントラクトが 4 件，NFT 型コントラクトが 5 件あることが分かった．以上より，this.balance による指標はあやしいコントラクトを発見する特徴として有効だと考えられる．

## 6. 考察

### 6.1 コントラクトハニーポットの追跡調査

5.3 節の分析において 2017 年 1 月〜2019 年 5 月の範囲でコントラクトハニーポットの被害が減少傾向にあるとは言えず，今後も脅威に成り得るという結果に至った．そこで，2019 年 5 月以降の期間を対象にコントラクトハニーポットの被害状況を分析する．

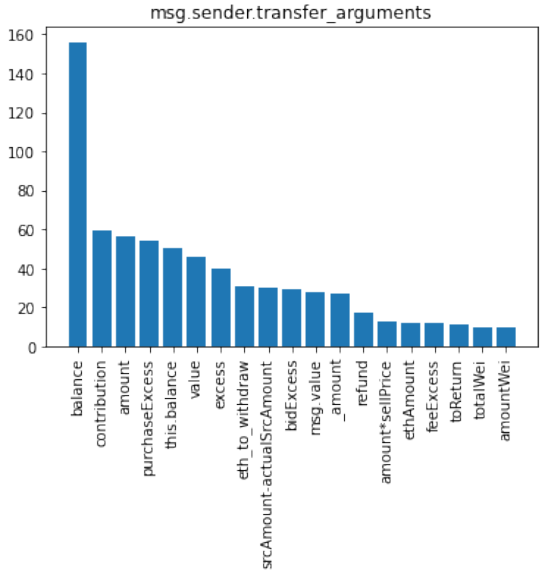


**図 6** 正規コントラクトにおける msg.sender.transfer 関数の引数

まず，2019 年 5 月以降で何らかの Solidity の関数を使用したコントラクトハニーポットを EtherScan 上のトランザクションの項目にある Method 列を目視で確認する．次に，イーサの送金処理やコントラクトの自己破壊などといった詳細な動作を捉える．

2019 年 5 月以降を分析期間として追跡調査を実施した結果を表 4 に示す．2019 年 5 月以降に関しては新たに 6 件の被害が確認でき，いずれもコントラクトハニーポットに騙されている．さらにその被害額の合計は約 2.11ETH であった．また，次の 2 つの行為が明らかになった．それは，コントラクトハニーポット設置者による管理行為（具体的にはウォレットにたまった被害額を引き出す・コントラクトを削除する），及びイーサリアムユーザがコントラクトに貯まっているイーサを引き出す行為である．

### 6.2 被害の有無の差について

本節では，特徴的である 3 種類のコントラクトハニーポット（HSU，ID，SESL）を扱う．特に，最も被害が大きかった HSU，1 件当たりの被害額が最大 7ETH だった ID，及び被害の突発性が確認され，他のコントラクトハニーポットと比較した時のトラップの特殊性（送金処理の引数が複数個用意されている・送金処理を独自で定義している・トラップの難解性）が確認できた SESL を対象とする．なお，本調査に用いたコントラクト数は 172 件である．特に，コントラクトハニーポットが攻撃に失敗した事例は 112 件であった．その理由は次のとおりである．

- 誰もアクセスしてこなかった．
- イーサリアムユーザが 0ETH を用いて確かめていた．
- 特に HSU において，コントラクト設置者がト

表 4　2019 年 5 月以降にコントラクトに何らかの動作があったコントラクト一覧

| トランザクションハッシュ（上位 5 桁） | コントラクトアドレス（上位 5 桁） | トラップドア名 | SolidityMethod | 挙動確認日時 | トランザクション送信者（上位 5 桁） | 被害額（ETH） |
|---|---|---|---|---|---|---|
| 0x456ca... | 0x68af0... | HSU | SetPwd | 2022/5/29 06:42 | 0x328eb... | 1.000001 |
| 0x61909... | 0x68af0... | HSU | SetPwd | 2022/5/29 06:36 | 0x328eb... | 1.0 |
| 0x06cd2... | 0x68af0... | HSU | CashOut | 2022/5/28 01:37 | 0x48088... | 1E-18 |
| 0x2b129... | 0x68af0... | HSU | SetPwd | 2022/5/28 01:33 | 0x48088... | 1E-18 |
| 0x54983... | 0x413c8... | US | play | 2022/05/23 09:10 | 0x48088... | 0.1 |
| 0x54983... | 0x413c8... | US | play | 2022/05/23 07:33 | 0x48088... | 0.11 |
| 0x42009... | 0x3e784... | ID | Kill | 2021/1/19 22:18 | 0x80028... | - |
| 0x5e179... | 0x3e784... | ID | Withdraw | 2021/1/19 22:16 | 0x80028... | - |
| 0xc036f... | 0x3bafb... | BD | Withdraw | 2021/1/19 21:54 | 0x125d6... | - |
| 0x53bf9... | 0x4dc76... | ID | TakeAll | 2021/1/19 21:43 | 0x80028... | - |
| 0xbfa13... | 0x96830... | US | Kill | 2020/12/28 22:09 | 0xfc8c5... | - |
| 0xa95ef... | 0x9f54d... | BD | Kill | 2020/9/20 11:14 | 0x4ef20... | - |
| 0x5098e... | 0xc0c7d... | ID | Withdrawfunds | 2019/6/23 0:52 | 0xa5b8f... | - |
| 0x1be0b... | 0x68af0... | HSU | CashOut | 2019/5/29 17:35 | 0x7d902... | - |

ラップの使い方を知らなかったあるいはトラップの設置を忘れたため，逆にイーサリアムユーザーにイーサーを窃取された

### 6.3　正規ユーザーの被害について

本節においても，特徴的である 3 種類のコントラクトハニーポット（HSU，ID，SESL）の 172 件を扱う．コントラクトの中身を確認すると，ゲームに模して被害者を誘うようなコントラクトが 124 件，あえて脆弱にみせたコントラクトが 48 件の 2 種類が存在した．特に前者は，ゲームのコントラクトであるため，攻撃者のみがコントラクトハニーポットにアクセスしてくるとは限らないので，そのコントラクトが脆弱なハニーポットかどうかに関係なく正規ユーザーがアクセスしてくる可能性があり，正規ユーザーの被害も考えられる．

## 7.　まとめ

本稿では，コントラクトハニーポットによる被害規模の把握に加えて，Solidity コードにおけるコントラクト名と msg.sender.transfer の引数をベースとしたコントラクトハニーポットのリスク分析を行った．コントラクト名・引数による分析においては，EtherScan 上で簡単に確認できるハニーポットのクラス名の特徴や注視すべき msg.sender.transfer 関数の引数の特徴を提示し，this.balance による未知コントラクトハニーポット検知の可能性を示唆した．また，トランザクションをベースとしたコントラクトハニーポットの被害有無の考察をするとともに，正規ユーザの被害についても分析した．

今後の展望としては，直近 1〜2 年分の Solidity コードを収集し，this.balance により新しいコントラクトハニーポットの存在の確認と被害規模と被害者の把握を行うことを検討している．

**謝辞**　本研究成果の一部は，JSPS 科研費 22H03588 の助成を受けたものである．

### 参考文献

[1] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[2] V. Buterin. A next-generation smart contract and decentralized application platform. white paper, 2014.

[3] R. Camino, C.F. Torres, M. Baden, and R. State. A data science approach for detecting honeypots in ethereum. IEEE ICBC 2020, pp.1 – 9. IEEE, 2020.

[4] A.K. Gogineni et al. Multi-class classification of vulnerabilities in smart contracts using awd-lstm, with pretrained encoder inspired from natural language processing. arXiv: 2004.00362, 2020.

[5] J.W. Liao, T.T. Tsai, C.K. He, and C.W. Tien. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. IOTSMS 2019, pp.458 – 465. IEEE, 2019.

[6] L. Luu et al. Making smart contracts smarter. CCS 2016, pp.254 – 269, 2016.

[7] C.F. Torres et al. The art of the scam: Demystifying honeypots in ethereum smart contracts. USENIX Security 19, pp.1591 – 1607, 2019.

[8] W. Chen et al. Honeypot Contract Risk Warning on Ethereum Smart Contracts. IEEE JCC 2020.

[9] T. Hu et al. Transaction-based classification and detection approach for Ethereum smart contract. Information Processing and Management 58 (2021) 102462, 2021.

[10] K. Hara, T. Takahashi, M. Ishimaki, K. Omote, Machine-learning Approach using Solidity Bytecode for Smart-contract Honeypot Detection in the Ethereum, IEEE QRS-C, 2021.

[11] “EVM”, https://github.com/CoinCulture/evmtools/blob/master/analysis/guide.md, September 20, 2021.