ID	脆弱性名	旧ID			説明	リスク						レイヤー	原因		SC特異性
						資産			プログラム						
		Vidal	Rameder	CWE		C	I	A	C	I	A		項目	詳細	
SC.1.1	Malicious Fallback Function	1.2	1B	685	call() 関数による関数呼び出し時に、呼び出し先のコントラクトで実行される fallback() 関数に悪性な挙動または不備が含まれる。また、fallback() 関数に限らず、あるコントラクトから任意の悪性な関数を呼び出すことができる脆弱性も含まれる。	-	-	-	-	○	△	SCプログラム内部	関数の挙動確認不足/誤認	関数シグネチャ（関数名と引数型）指定のない状態で call() 関数を使用している、または任意のコントラクトの関数を呼び出せる仕様になっているため。	○
SC.1.2.1	Improper Check of External Call Return Value	1.3.1	10A	252	外部コントラクトの関数実行結果（返り値）を検証せずに使用すると、予期せぬ挙動となる可能性がある。	-	-	-	-	○	○	SCプログラム内部	外部ライブラリの挙動確認不足	外部コントラクトの関数実行結果の検証が不十分なため。	
SC.1.2.2	Improper Exception Handling of External Calls	1.3.2	-	703	外部プログラムやライブラリで発生した例外・エラーが呼び出し元のコントラクトに波及し、コントラクト実行が停止する。	-	-	-	-	○	△	SCプログラム内部	関数の挙動確認不足/誤認	外部プログラムやライブラリ、または呼び出し元コントラクトにおける例外処理が不適切・不十分であるため。	
SC.1.2.3	Improper Check of Low-Level Call Return Value	1.3.3	3A	372	暗号資産を送金できる call.value() 関数や call() 関数を使用する際、その返り値に依存する分岐プログラムにおいて、送金失敗時の実装に不備がある。	-	-	-	-	○	△	SCプログラム内部	関数の挙動確認不足/誤認	call.value() 関数の返り値に依存する分岐プログラムを適切に実装していないため。	○
SC.1.3	Improper locking during external calls	1.4	-	667	関数や値のロック処理が不適切であるとデッドロック状態になり、コントラクト実行が停止する。	-	-	-	-	-	○	SCプログラム内部	ビジネスロジックエラー	ノードやコントラクトが同時アクセスできる関数や値においてロック処理が不適切であるため。	
SC.1.4	Interoperability issues with other contracts	1.5	8F	843	コンパイラバージョンの異なる複数のコントラクトを使用している場合に、新しいバージョンのコントラクトで実行した古いバージョンのコントラクト機能が予期せぬ挙動を示す可能性がある。特に、assembly() 関数やstaticcall() 関数はバージョンによって挙動が変わるため注意が必要である。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	使用している複数のコントラクトの間でコンパイラバージョンが異なっている。	○
SC.1.5	Delegatecall to Untrusted Callee	1.6	1E	940	呼び出し先のストレージ（変数等のコントラクトの状態）を変える call() 関数と勘違いして、呼び出し元のストレージを変える delegatecall() 関数やcalldata() 関数を使用し、予期せぬ挙動を示す可能性がある。	-	-	-	△	○	△	SCプログラム内部	関数の挙動確認不足/誤認	delegatecall() 関数やcalldata() 関数を挙動を十分に理解せずに使用しているため	○

SC.1.6	Cross Channel Invocation	1.8	-	435	チャネルという仮想ネットワークでコントラクト同士互いに呼び出せる Hyper Fabric において、コントラクトAがコントラクトBを呼ぶチャネルと、BがAを呼ぶチャネルが異なっている場合、予期せぬ挙動になる可能性がある。	-	-	-	-	○	-	SCプログラム内部	固有のブロックチェーンで起こる仕様確認不足	HyperFabricブロックチェーンを使っている場合、コントラクトを互いに呼び出す際に異なるチャネルで呼び出しているため。	○
SC.2.1.1	Improper Use of Exception Handling Functions	2.1.1	3B	248	GASが足りない場合やオーバーフロー発生時などのエラーを適切に処理しておらず、予期せぬ挙動になる。	-	-	-	-	△	○	SCプログラム内部	ビジネスロジックエラー	例外処理が不適切・不十分であるため。	
SC.2.1.2	Improper Exception Handling in a Loop	2.1.2	5B	400	ループ処理等により実行するプログラム文が過剰に多い場合、GAS代やトランザクションサイズが増加し、最悪の場合、ブロックチェーン上に取り込まれず GAS 代だけ消費させられる。	-	-	△	-	-	△	SCプログラム内部	ブロックチェーン層の挙動確認不足	実行されるステートメントが過剰に多いため。	○
SC.2.1.3	Incorrect Revert Implementation in a Loop	2.1.3	4E	400	コントラクトの実行失敗時にリバート処理（実行前の状態へ戻す処理）が適切でないと、コントラクトの状態や実行結果に不整合が生じる。例えば、ループ処理におけるエラー処理が不適切で一部が実行され残りは実行されなかった状態になる。	-	-	△	-	○	△	SCプログラム内部	ビジネスロジックエラー	コントラクトの実行失敗時におけるリバート処理の実装が不十分なため。	○
SC.2.2.1	Missing Thrown Exception	2.2.1	-	474	暗号資産の送金結果として返り値のある transfer() 関数において、送金失敗時のエラーを適切に発出しないと送金できなかった理由（関数実行できなかった理由）が不明瞭になり誤解が生じる可能性がある。	-	-	△	-	○	-	SCプログラム内部	ビジネスロジックエラー	適切な例外を発出していないため。	
SC.2.2.2	Extraneous Exception Handling	2.2.2	9D	474	推奨実装 (Tokenを扱うAPIなど) に対する追加実装が、適切に実行されていない可能性がある。	-	-	-	-	○	○	SCプログラム内部	ビジネスロジックエラー	追加実装による処理が不適切・不十分なため。	
SC.3.1	Improper Check on Transfer Credit	4.1	-	391	送金が正しく完了したのか、送金先は意図しているアドレスなのかなどの検証が行われずに、意図していない相手への送金や送金が失敗したのにもかかわらず成功した場合のふるまいになってしまう。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	送金先アドレスや送金成功可否の検証を行っていないため	○
SC.3.2	Unprotected Transfer Value	4.2	6D	400	資産を取り扱う関数、特に資産を送金する関数のアクセス制御が不適切だと、資産を盗まれる可能性がある。	○	-	○	-	△	△	SCプログラム内部	関数実行の権限管理不足	ETHを送金する関数が他の人でも使える状態であるため。	○

SC.3.3	Wrong use of Transfer Credit Function	4.3	-	840	send() 関数は送金の成否をbool値で返す仕様であり、transfer()関数は送金失敗時にエラーを送出する仕様である。この異なる仕様を知って実装していないと予期せぬ挙動になる可能性がある。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	send() 関数とtransfer() 関数の挙動の違いを理解していないため。send() 関数は送金の成否をbool値で返し、transfer() 関数は送金失敗時エラーを送出する。	○
SC.3.4	Missing Token Issuer Verification	4.4	6B	304	EOSIOブロックチェーンに関連しており、送金機能によってはあるチケット代を払わずに暗号資産を獲得できてしまう可能性がある。	△	-	○	-	○	-	SCプログラム内部	固有のブロックチェーンで起こる仕様	EOSIOブロックチェーンでは誰でも任意の名前でトークンを発行できるため。	
SC.3.5	Missing Token Verification of Exchange	4.5	6B	306	送金やトークン送信時の送信先の検証が不十分であり、transfer() /transferFrom() 関数が正しく実行されずに送信できない場合が該当する。	△	-	○	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	safeTransferFrom() 関数を使っておらず、推奨されていないtransfer() 関数やtransferFrom() 関数の使用をしているため。	
SC.3.6	Fake Notification	4.6	-	223	EOSIOブロックチェーンによっておこる脆弱性であり、EOS通知が偽造できてしまうため注意が必要である。	-	-	-	-	○	-	SCプログラム内部	固有のブロックチェーンで	EOSIOブロックチェーンにおいてeosponserの通知を検証しないとEOS通知は偽造できてしまうため。	○
SC.4.1.1	Missing Constructor	5.2.1	-	477	コンストラクタはデプロイ時に一回だけ実行される関数であり、コンストラクタ内で初期設定を行うことが多い。そのコンストラクタが無いと初期設定がうまく行えない可能性がある。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	コンストラクタの仕組みを理解していないため。	○
SC.4.1.2	Wrong Constructor Name	5.2.2	10C	665	コンストラクタは"constructor"という名前の関数を実装するもしくはコントラクト名と同じ名前の関数を実装することで実現されるが、コンストラクタとしたい関数名がコントラクト名とは違った名前であるとコンストラクタではなく通常の関数として扱われてしまう。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	コンストラクタとしたい関数名がコントラクト名と異なっているもしくは"constrctor"という名前で実装していないため	○
SC.4.1.3	Missing variable initialization	5.2.3	-	908	変数の初期化をしていないと、予期せぬ値が格納される。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	変数の初期値を設定しておらず、開発者の意図しない値が格納されているため。	
SC.4.1.4	Extraneous Field Declaration	5.2.5	-	665	構造体内のフィールドにアクセスできてしまうと、同期が難しくノード間で矛盾が生じる可能性がある。	-	-	-	-	○	-	SCプログラム内部	固有のブロックチェーンで	HyperFabricでのchaincode構造体でフィールド宣言をしているため。	○
SC.4.1.5	Hardcoded Address	5.2.6	-	1052	関数実行者変数 msg.sender などを使わずに直接プログラムにアドレスを書いている場合、契約プログラムをアップデートしたい際にアドレスを変更することができなくなってしまう。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	アドレスを直書きしているため。	○

SC.4.2.1	Function Call with Wrong Arguments	5.4.3	-	116	右から左へのオーバーライド制御文字が引数として渡されると、引数の順序が逆になって関数が実行されてしまう。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	Unicodeの使用制限などを設けていないことや、引数のチェックを行っていないため。	
SC.4.3	Wrong class inheritance order	5.5	8G	696	別コントラクトを複数継承するとき、順番を間違えると意図しない挙動になる。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	テストや他の開発メンバーなどによる確認の不備があり、間違った順序で継承を行っているため。	
SC.4.4.1	Missing return type on function	5.6.1	-	694	関数の戻り値の型を宣言しないと、意図した型とは別の型で格納される危険性がある。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	戻り値の型を宣言していないため。	
SC.4.4.2	Function return type mismatch	5.6.2	-	694	返り値に間違った型を指定してしまうと、予期せぬ挙動となる。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	関数の返り値の型と受け取る変数の型が異なるため。	
SC.4.4.3	Parameter type mismatch	5.6.3	-	704	関数やインタフェースの引数の型を間違えると予期せぬ挙動になる。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	関数の引数の型と呼び出し時の変数の型が異なるため。	
SC.4.4.4	Missing type in variable declaration	5.6.4	-	695	変数の型を宣言しないと予期せぬ挙動になる。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	変数の型を宣言していないため。	
SC.4.4.5	Wrong Type of Function	5.6.6	-	668	ブロックチェーン（storage）のデータ取得可能・編集不可な view 型の関数と、どちらもできない pure型の関数の仕様を混同し、pure 型でデータを取得しようとするとスマートコントラクトの実行はエラーとなる。	-	-	-	-	○	-	SCプログラム内部	関数実行の権限管理不足	関数修飾子 view および pure の仕様を正しく理解していない/誤設定しているため。	○
SC.4.4.6	Non-Identifiable Order in Map Structure Iteration	5.6.7	-	462	Hyper FabricブロックチェーンではGolangにてSCを実装できるが、Golang言語のMapはキーとバリューのペアが一意ではない可能性がある。	-	-	-	-	○	-	SCプログラム内部	固有のブロックチェーンで起こる仕様	Go言語Mapはキーと値のペアの順序が一意でないことがあるため	○
SC.4.5.1	Unreachable Payable Function	5.7.1	4A	561	コントラクトが暗号資産を受理できる機能を持つが転送できる機能が無い場合、Ownerなどがこのコントラクトから資産を引き出すことができずにロックされてしまう。	○	-	○	-	△	-	SCプログラム内部	ビジネスロジックエラー	コントラクトから資産を引き落とす関数を実装していないため。	
SC.4.5.2	No effect code execution	5.7.2	8J	1164	未使用の関数や、実行前後で何も変わらないような意味の無いコードは開発者の混乱の原因となる。また、Gas代が増える可能性がある。	-	-	△	-	△	△	SCプログラム内部	ビジネスロジックエラー	テストや他の開発メンバーなどによる確認の不備により、意味のないコードが存在しているため。	
SC.4.5.3	Unused Variables	5.7.3	8J	563	未使用変数を無くし、予想してない状況にならないようにすべき。余計なmemoryを使用しGas代が高くなる可能性がある。	-	-	△	-	△	△	SCプログラム内部	ビジネスロジックエラー	テストや他の開発メンバーなどによる確認の不備により、使用していない変数があるため。	

SC.4.5.4	Inefficient Operation Sequence	5.7.4	-	1281	プログラムや実行内容が最適化されていないと、GAS代が増加してしまう。	-	-	△	-	-	△	SCプログラム内部	コンパイラバージョンの確認不足	コンパイラバージョンが古かったり、適切にリファクタリングされておらず、プログラムが最適化されていないため。	○
SC.4.6.1	Undetermined Program version Prevalence	5.8.1	9C	664	コンパイラバージョンを明示しないと、意図しないバージョンでコンパイルされ、不具合が生じる可能性がある。	-	-	-	-	○	-	SCプログラム内部	コンパイラバージョンの確認不足	コンパイラバージョンが指定されていないため。	○
SC.4.6.2	Outdated Compiler Version	5.8.2	9B	937	古いバージョンのコンパイラを使用すると、予期せぬ挙動やバグが発生する可能性がある。	-	-	-	-	○	-	SCプログラム内部	コンパイラバージョンの確認不足	古いバージョンのコンパイラを使用しているため。	○
SC.4.6.3	Use of Deprecated Functions	5.8.3	8D	477	非推奨の関数の利用(suicide,callcode,sha3など)を避けるべき。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤	当該関数を使用しているため	○
SC.4.7	Inadequate Data Representation	5.9	-	1093	桁数の多い値を直接プログラムに記載すると、桁数の間違いや可読性を損ねる。	-	-	△	-	○	-	SCプログラム内部	ビジネスロジックエ	直書きしているためや10**16などの表記にしていないため。	
SC.4.8.1	Wrong Function Modifier	5.10.1	5D	400	関数の intenal call は必要ないにも関わらず、関数修飾子 external ではなく public を設定してしまうと、GAS代の高いコードとなってしまう。	-	-	-	-	○	△	SCプログラム内部	GAS代の考慮不足	関数修飾子 public および external を適切に設定していないため。	○
SC.4.8.2	Missing Constant Modifier in Variable Declaration	5.10.2	-	710	定数となる変数に修飾子 constant を指定しないと、GAS代の高いコードとなってしまう。	-	-	△	-	-	△	SCプログラム内部	GAS代の考慮不足	定数の変数に constant を設定していないため。	○
SC.4.8.3	Missing Visibility Modifier in Variable Declaration	5.10.3	-	710	modifierのデフォルトはinternalで派生コントラクトからアクセスが許可されることに留意。	-	-	-	○	-	-	SCプログラム内部	関数実行の権限管理不足	適切な修飾子を付けていないため。	○
SC.4.9	Redundant Functionality	5.11	-	1041	可読性の低いコードや大量の分岐や関数呼び出しを行うなど冗長に書くと保守が困難になる。	-	-	-	-	△	-	SCプログラム内部	ビジネスロジックエラー	意味のないコードや適切な表現でないコードがあるため。	
SC.4.10.1	Use of Same Variable or Function Name in Inherited Contract	5.12.1	8H	1109	親コントラクトと同じ変数名を子コントラクトでも使う場合、変数宣言をしないと親コントラクトの変数値が変更されてしまう。	-	-	-	△	○	△	SCプログラム内部	ビジネスロジックエラー	継承しているコントラクトで親プログラム内にある同じ変数名を使用しているため。	
SC.4.10.2	Variables or Functions Named After Reserved Words	5.12.2	-	1109	予約語の関数名を宣言してしまうと、衝突を起こし予期せぬ挙動となる。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	Solidity言語における予約語において、予約語とは知らずに同じ名前の変数や関数を実装してしまっているため。	○
SC.4.10.3	Use of the Same Variable of Function Name in a Single Contract	5.12.3	-	1109	関数や変数名を同じにすると予期せぬ挙動になる。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	変数と関数が同じ名前になっているため。	

SC.4.11.1	Write to Arbitary Storage Location	5.13.2	8E	123	配列について、push, popやインデックス指定による格納値の変更などにより配列内の値が任意の値に変更できてしまう可能性がある。	-	-	-	△	○	-	SCプログラム内部	関数実行の権限管理不足	配列の更新を他者が行う際に、インデックスが範囲内であるのかの確認を行っていないため。また、(配列).lengthをインクリメントやデクリメント、値指定を	
SC.4.11.2	Read from Arbitrary Storage Location	5.13.3	-	127	Hyper Fabricブロックチェーンではgolangにて SCを実装できるが、Golang言語の配列でのインデックスがオーバー/アンダーフローした際に、その配列以外の変数の値を読み取ることができてしまう場合がある。	-	-	-	○	-	-	SCプログラム内部	関数実行の権限管理不足	Golangの配列のインデックスがオーバー/アンダーフローが起こるような指定をしているため。	○
SC.4.12	Use of Malicious Libraries	5.14	2E	829	悪意のある外部ライブラリを使用している場合が該当する。	-	-	-	-	○	○	SCプログラム内部	関数実行の権限管理不	外部ライブラリに関する調査が足りず悪性なものかの判断が出来ていないため。	
SC.4.13	Typographical Error	5.15	8B	480	「+=」を「=+」にするなどのタイポやコーディングミス	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエ	テストや他の開発メンバーなどによる確認の不備があるため。	
SC.4.14	Wrong Logic	5.16	-	840	アルゴリズムなどのロジックが間違っていると予期せぬ挙動となる。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエ	ロジックの確認が不足しているため。	
SC.4.15	Wrong moment for token generation	5.17	-	179	トークンの生成プロセスがそのトークンを受け取るノードにのみ依存している場合、他者に任意のトークンが作成され価格を操作される危険性がある。	-	○	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	トークンの生成プロセスが開発者の承認が無く、他者がトークンのメタデータなどを任意の値に設定できてしまうような受け取り手にのみ依存しているため。	○
SC.5.1.1	Incorrect Function Call Order	6.1.2	-	362	複数の関数実行をするための順序がある場合、実行順序を制限しないと予期せぬ挙動となる。	-	-	-	-	○	-	SCプログラム内部	ビジネスロジックエラー	関数実行順序に制限がなく、任意の関数を実行できてしまうため。	
SC.5.1.2	Improper Locking	6.1.3	1C	667	アカウントやコントラクトの残高を条件式に組み込むと、攻撃者がそのアカウントやコントラクトに送金することにより条件式の結果を制御できてしまう。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	アカウント/コントラクトの残高を条件式に使用しているため。	○
SC.5.1.3	Exposed State Variables	6.1.7	-	200	コントラクトの状態変数へのアクセス制御が不適切だと、攻撃者がその状態変数を更新することによりコントラクトに影響を与えられてしまう。	-	-	-	○	○	-	SCプログラム内部	ビジネスロジックエラー	コントラクトの状態変数へのアクセス制御が不適切なため。	○
SC.5.1.4	Wrong Transaction Definition	6.1.8	-	1251	EOSIOブロックチェーンによっておこる脆弱性であり、インラインアクションをロールバックする機能を悪用することが可能である。	-	-	-	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	インラインアクションがBCのトランザクションを拒否できることを知って開発していないため。	○
SC.5.1.5	Stopped Process due to Nodes Doing Nothing	-	-	15	コントラクトの実行フローが任意のノードのみに依存している場合、それらノードが悪意を持ってフローを止めるとコントラクト実行が停止する。	-	-	-	-	-	○	SCプログラム内部	ビジネスロジックエラー	開発者が管理するノード等の他の特定のノードもコントラクトの実行フローを進められる設定や実行時間に制限設定を適用していないため。	○

SC.5.2.1	Improper Input Validation	6.2.1	9A	20	関数の入力値を適切に検証しないと、後続の処理が予期せぬ挙動となる可能性がある。例えば、Solidity では短いアドレスに対してゼロデータでパディングするため、後続の処理で不整合が生じる可能性がある。	△	-	△	-	○	○	SCプログラム内部	値の精査不足	入力値の検証が不適切・不十分であるため。	
SC.5.2.2	Extraneous Input Validation	6.2.2	-	573	require 文などを用いて関数入力値のエラーを検証できるが、True となるべき検証が False となり、コントラクト実行が停止する。	-	-	-	-	△	○	SCプログラム内部	ビジネスロジックエラー	True となるべきプログラムの検証が False となっているため。	
SC.6.1.1	Integer Underflow	7.1.1	-	191	整数型の下限を超える値を扱うとアンダーフローを引き起こす。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	数値の検証を行っていないため。	
SC.6.1.2	Integer Overflow	7.1.2	-	190	整数型の上限を超える値を扱うとオーバーフローを引き起こす。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	数値の検証を行っていないため。	
SC.6.2.1	Divide by zero	7.2.1	-	369	整数を0で割ってしまうとゼロ除算のエラーを引き起こす。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	割算分母が0になっているため。	
SC.6.2.2	Integer Division	7.2.2	7B	682	整数型の値の算術処理において、割り算を掛け算より先に適用すると誤差を誘発する。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	割り算による商の誤差を認識していないため。	
SC.6.3.1	Truncation Bugs	7.3.1	-	197	uint256型からuint8型へなど、サイズの小さい型へ変数をキャストすると値の精度が低下し、予期せぬ値となる可能性がある。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	データサイズの小さな型へ変数をキャストしているため。	
SC.6.3.2	Signedness Bugs	7.3.2	-	195	符号付き整数型から符号なし整数型へ値を変換する場合、予期せぬ値となる可能性がある。	-	-	-	-	○	-	SCプログラム内部	値の精査不足	符号の異なった型に代入しているため。	
SC.7.1.1	Wrong Caller Identification	8.1.1	6A	1126	EOAアカウントおよびコントラクトのアドレスどちらも格納される msg.sender 変数と、EOAアカウントのアドレスが格納される tx.origin 変数の仕様の違いに留意しないと予期せぬ挙動になる恐れがある。 例えば、関数Aから関数Bを呼び出し関数B内で msg.sender と tx.origin が定義される場合、msg.sender は関数Aを含むコントラクトアドレスが格納され、tx.origin は関数Aを呼び出したEOAアカウントのアドレスが格納される。	△	-	△	-	○	-	SCプログラム内部	関数の挙動確認不足/誤認	msg.sender と tx.origin の仕様を理解していないため。	○
SC.7.1.2	Owner Manipulation	8.1.2	-	732	特別な権限を付与しているアドレス型の変数 owner の値が変更可能であると、予期せぬ挙動となる可能性がある。	△	-	△	-	○	-	SCプログラム内部	関数実行の権限管理不足	アドレス型の owner 変数が変更可能なため。	○

SC.7.1.3	Missing verification for program termination	8.1.3	6C	1082	selfdestruct() 関数のアクセス制御が不適切だと、悪意のあるアカウントに実行されコントラクトが使えなくなる。	-	-	-	-	-	○	SCプログラム内部	関数実行の権限管理不足	selfdestruct() 関数のアクセス制御が不適切であるため。	○
SC.7.2.1	Incorrect Verification of Cryptographic Signature	8.3.1	6E	347	署名方法が不適切であると、誰でも正しく署名できる等の不正行為ができる可能性がある。	-	-	△	○	-	-	SCプログラム内部	ビジネスロジックエラー	署名方法が不適切であるため。	
SC.7.2.2	Improper Check against Signature Replay Attacks	8.3.2	2F	347	リプレイ攻撃可能な署名検証では、悪意のあるクライアントが正当なトランザクションのメッセージハッシュを取得することにより、同じ署名を使用して正当なクライアントになりすましできる可能性がある。	-	-	-	○	-	-	SCプログラム内部	ビジネスロジックエラー	既に利用された値でも認証できるリプレイ攻撃が可能であるため。	○
SC.7.2.3	Improper Authenticity Check	8.3.3	6B	345	変数や関数のアクセス修飾子が不適切だと、保護すべき変数や関数が他者に利用されてしまう可能性がある。	-	-	-	○	-	-	SCプログラム内部	関数実行の権限管理不足	保護すべき変数や関数の修飾子が private などになっていないため。	
SC.7.2.4	Incorrect Argument Encoding	8.3.4	-	294	衝突をよく起こす abi.encodePacked() 関数は、入力となる配列の要素順に関係なく同じ値が出力されてしまい、予期せぬ挙動となる可能性がある。特に、本関数を用いた認証は正しく行えない可能性がある。	-	-	-	○	-	-	SCプログラム内部	関数の挙動確認不足/誤認	ハッシュ関数やabi.encodePacked() 関数の仕様や挙動を理解していないため。	○
VM.1.1.1	Unsafe Credit Transfer	1.1.1	1A	841	call() 関数が呼び出し元コントラクトの fallback() 関数を実行する仕様のため、fallback() 関数が再帰的に call() 関数を呼び出すループ構造の実装（リエントランシーと呼ばれる）になっていると、call() 関数で送金処理を行うと、コントラクトに紐づく暗号資産が尽きるまで送金してしまう。	○	-	○	-	○	-	EVM	関数の挙動確認不足/誤認	call() 関数を用いて暗号資産を送金しており、かつ fallback() 関数により再帰的に当該 call() 関数が呼び出されてしまう実装になってしまっているため。	○
VM.1.1.2	Unsafe System State Changes	1.1.2	-	941	送金ではない処理に対するリエントランシー（cf. VM.1.1.1）により、意図しないコントラクトの状態に陥り、コントラクトの性能・可用性低下を招く。	-	-	-	-	○	-	EVM	関数の挙動確認不足/誤認	fallback() 関数により再帰的に当該 call() 関数が呼び出されてしまう実装になってしまっているため。	○
VM.2.1	Improper Gas Requirements Checking	3.1	5E	691	GAS 不足でコントラクトの実行が途中で停止してしまう。	-	-	△	-	-	△	EVM	GAS代の考慮不足	コントラクトの実行に必要な GAS代 の総額を確認していない/誤認識しているため。	○
VM.2.2	Call with hardcoded gas amount	3.2	9E	655	ハードコードされた GAS 量に基づき GAS 代を支払うコントラクトを実行すると、ハードフォーク等により必要な GAS 量が増加した場合に実行できなくなる。	-	-	-	-	△	△	EVM	GAS代の考慮不足	GAS代の支払いにハードコードされた GAS 量を使用しているため。	○

VM.3.1.1	Uninitialized Storage Variables	5.2.4	10B	453	ストレージ由来の状態変数を適切に初期化しないと、ストレージスロットのアドレス衝突を起こし、予期せぬ挙動となる可能性がある。	-	-	-	-	○	-	EVM	値の精査不足	状態変数を適切に初期化していないため。。	○
VM.3.2.1	Wrong Function Call	5.4.1	-	328	同じ関数シグネチャ（関数名や引数型）を持った関数が複数存在すると、意図していない関数を実行してしまう可能性がある。	-	-	-	△	○	-	EVM	関数の挙動確認不足/誤認	コントラクト内に同じ関数シグネチャの関数が複数存在するため。	
VM.3.2.2	Wrong Selection of Guard Function	5.4.2	3D	670	基本的に開発時のテストで使用する assert 文を外部からの入力データやエラー検証に使用していることにより、意図せずコントラクトの実行が停止する可能性がある。	-	-	-	○	○	○	EVM	関数の挙動確認不足/誤認	基本的に開発時のテストで使用する assert 文と、外部からの入力データ検証で使用する require 文を混合してしまっているため。	○
VM.3.3.1	Wrong Type in Variable Declaration	5.6.5	5C	789	メモリサイズを余計に使用する型の変数を使用すると、GAS代も余計に消費してしまう。	-	-	△	-	-	△	EVM	GAS代の考慮不足	適切な変数型を使用していないため。	○
VM.3.4.1	Stack-based Buffer Overflow	5.13.1	-	121	EVM の実行スタックをオーバーフローさせることができてしまうと、特定のリソースにアクセスしたり、制御変数を上書きできてしまったりする。	-	-	-	-	○	-	EVM	コンパイラバージョンの確認不足	古いバージョンのEVMを使用しているため。（最新バージョンではオーバーフローは発生しない）	○
BC.1.1	Bad Randomness	5.1	2C	330	block.timestamp, blockhash, block.difficultyなどのブロックチェーン情報はマイナーが操作可能であるため、これらの情報を乱数生成に利用すると乱数が改ざんされる恐れがある。	-	-	-	○	-	-	ブロックチェーン	ブロックチェーン層の挙動確認不足	マイナーが操作可能なブロックチェーン情報を乱数生成に利用しているため。	○
BC.2.1.1	Incorrect Use of Event Blockchain variables for Time	6.1.1	2A	829	コントラクトプログラムの制御がブロックチェーン情報に依存している場合、BC.1.1と同じ理由によりマイナーによる改ざんを受ける可能性がある。	-	-	-	-	○	-	ブロックチェーン	ブロックチェーン層の挙動確認不足	マイナーが操作可能なブロックチェーン情報をコントラクトプログラムの制御に利用しているため。	○
BC.2.1.2	Transfer Pre-Condition Dependent on Transaction Order	6.1.4	2B	364	イーサリアムのようにブロック単位で複数のトランザクションが処理されるブロックチェーンにおいては、処理されるトランザクションの順番が、必ずしもノードに送信された順番になるとは限らない。	-	-	-	-	○	-	ブロックチェーン	ブロックチェーン層の挙動確認不足	トランザクションの実行順序で結果が変わってしまう仕様であるため。	○
BC.2.1.3	Transfer Amount Depending on Transaction Order	6.1.5	-	364	ブロック内のトランザクションの順序により、送金額を決定する変数の値が予期せぬ値に変更され、本来送金したい額とは異なる金額で送金する恐れがある。		-	-	-	○	-	ブロックチェーン	ブロックチェーン層の挙動確認不足	トランザクションの実行順序で結果が変わってしまう仕様であるため。	○

BC.2.1.4	Transfer Recipient Depending on Transaction Order	6.1.6	-	364	ブロック内のトランザクションの順序により、送金処理が発生する前に送金先アドレスが格納されている変数などの値が変更され、別のアドレスに送金する恐れがある。	-	-	-	-	○	-	ブロックチェーン	ブロックチェーン層の挙動確認不足	トランザクションの実行順序で結果が変わってしまう仕様であるため。	○
BC.3.1.1	Exposed private data	8.2.1	2D	767	Solidity等においてprivateで定義された変数の値などはBC上では完全に不可視になるわけではなく、トランザクションの検証を担当するマイナー等からその変数の値等確認することができる。	-	-	-	○	-	-	ブロックチェーン	ブロックチェーン層の挙動確認不足	privateで定義された変数の値はBC上で完全に不可視にはならないということを理解していないため。	○
OC.1.1.1	Unsafe External Library Call	1.7.2	2E	829	ブロックチェーン外の外部ライブラリを呼ぶことで予期せぬ挙動になる可能性がある。	-	-	-	△	○	-	外部ライブラリ(オンチェーン)	外部ライブラリの挙動確認不足	外部ライブラリの内部挙動を理解していないため。	
OC.2.1.1	Dirty Reads	5.3.2	-	1100	Hyperledger Fabricではread/writeの競合が発生し同一トランザクション内でクエリが更新前値を返すことがあり、予期せぬ挙動になる可能性がある。	-	-	-	-	○	-	外部ライブラリ(オンチェーン)	固有のブロックチェーンで起こる仕様	Hyperledger Fabricはread/writeの一貫性をサポートしていないため、read/writeの競合が発生する挙動を把握せずに実装をしたため。	
OC.3.1.1	Dependency on External State Data on blockchain	8.2.2	-	642	コントラクトが管理や生成していないブロックチェーン上ではない外部データに依存している場合、その外部データの制約(生成条件や外部環境等)に依存することになり、コントラクトの実行結果が予期しないものになる可能性がある。	-	-	-	-	○	-	外部ライブラリ(オンチェーン)	外部ライブラリの挙動確認不足	コントラクトが制限できない外部データに依存した実装により、外部データの制約に依存した実行結果になるため。	
OF.1.1.1	Unsafe External Web Service Call	1.7.1	-	15	外部のウェブサービスを利用する際ノード間で異なる値が返される可能性があり、その挙動を理解していないと予期せぬ結果となる可能性がある。	-	-	-	-	○	-	外部ライブラリ(オフチェーン)	外部ライブラリの挙動確認不足	外部ウェブサービスはブロックチェーン上にはないものであるため、各ノードに同じ値を返すかどうか保証されているか不明である。その点を理解せずに実装し	
OF.1.1.2	Unsafe External Command Execution	1.7.3	-	829	外部コマンドに頼った実行はノード間で異なる結果になることがあり、その挙動を理解していないと予期せぬ結果となる可能性がある。	-	-	-	-	○	-	外部ライブラリ(オフチェーン)	外部ライブラリの挙動確認不足	外部コマンドは各ノードに同じ値を返すかどうか保証されているか不明であり、その点を理解せずに実装しているため。	
OF.1.1.3	Unsafe External File Access	1.7.4	-	829	外部ファイルアクセスはノード間で異なる結果になる可能性があり、その挙動を理解していないと予期せぬ結果となる可能性がある。	-	-	-	-	○	-	外部ライブラリ(オフチェーン)	外部ライブラリの挙動確認不足	外部ファイルへのアクセスは各ノードに同一の結果となるか保証されているか不明であり、その点を理解せずに実装しているため。	

OF.2.1.1	Phantom Reads	5.3.1	-	1100	Hyperledger Fabricが提供するGetPrivateDataQueryResultメソッド等は検証フェーズで再実行されないため、検証フェーズの間に同期されていないノードから古い情報に基づいて処理を行う可能性がある。	-	-	-	○	-	-	外部ライブラリ(オフチェーン)	固有のブロックチェーンで起こる仕様確認不足	Hyperledger Fabricにおいてファントムリード問題を検知できないgetPrivateDataQueryResultやgetQueryResult等のメソッドを利用しているため。	
OF.3.1.1	Dependency on External State Data on blockchain	8.2.2	-	642	コントラクトが管理や生成していないブロックチェーン上の外部データに依存している場合、その外部データの制約(生成条件や外部環境等)に依存することになり、コントラクトの実行結果が予期しないものになる可能性がある。	-	-	-	-	○	-	外部ライブラリ(オフチェーン)	外部ライブラリの挙動確認不足	コントラクトが制限できない外部データに依存した実装により、外部データの制約に依存した実行結果になるため。	