MINERY REPORT

Evaluación de Riesgos de Seguridad

Informe Ejecutivo y Técnico sobre Smart Contract

CriptoVision, noviembre 2021

C/ Antonio Suárez, 10
Alcalá de Henares 28802 Madrid
mineryreport.com

Detalles del Cliente	
Cliente	CriptoVision LLC
CIF/NIF	
Email de Contacto	cesar.patino@criptovision.com
Persona de Contacto	Cesar Patiño

Detalles del Documento	
Tipo	Reporte de Vulnerabilidades
Título	Evaluación de Riesgos de Seguridad
Descripción	Informe Ejecutivo y Técnico sobre Smart Contract
Clasificación	Confidencial
Fecha de Creación	noviembre 2021

Historial de Cambios			
Fecha	Versión	Autor	Descripción
17/11/21	1.0	Minery Report	Informe

COPYRIGHT

Este documento contiene información confidencial cuyo propietario es **CriptoVision**, quien tiene los derechos de copyright. Como tal está sujeto a secreto profesional, por lo que cualquier distribución, reproducción o divulgación de dicha información por cualquier medio está prohibida, sin autorización previa por escrito de **CriptoVision.**

Este documento no puede ser utilizado para otro propósito distinto por el que fue creado y se deben adoptar todas las medidas razonables que aseguren la confidencialidad de la información proporcionada en su contenido.

Este documento y cualquier otra información de naturaleza confidencial que pueda ser provista por **Minery Report** en el curso del proceso de ejecución del servicio están y estarán cubiertos por acuerdos de no divulgación entre **Minery Report** y **CriptoVision.**

Fecha de emisión del reporte: 17 de noviembre de 2021

C/ Antonio Suárez, 10
Alcalá de Henares 28802 Madrid
mineryreport.com

CONTENIDO

INFORME EJECUTIVO

Alcance del servicio

El presente documento es un reporte acerca de los resultados de la revisión de código y auditoría de un Smart Contract. Por tanto, el alcance de dichos servicios se limita al Smart Contract provisto por el cliente en el repositorio **https://github.com/criptovision/CRIPTOVISION-NEW-TOKEN.** En concreto, se hará referencia al fichero "**CriptoVision.**sol", correspondiente al commit "**cf9c5a6ae57cdafd5de91184e94ce7406e6f2a6c**", de la rama "**main**".

Pruebas Realizadas

Las pruebas a realizar durante los procesos de revisión y auditoría se englobarán dentro del esquema **SWC Registry** (Smart Contract Weakness Classification and Test Cases), que emplea la terminología y estructura del esquema CWE (Common Weakness Enumeration) de cara a proporcionar un estándar de identificación y clasificación de vulnerabilidades, así como definir un lenguaje común para describir debilidades de arquitectura y diseño del código y mejorar el rendimiento de las herramientas de análisis de seguridad.

En la actualidad, el esquema SWC define las siguientes:

ID	Título	CWE Asociado
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code

SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality
SWC-126	Insufficient Gas Griefing	CWE-691: Insufficient Control Flow Management
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature
SWC-116	Block values as a proxy for time	CWE-829: Inclusion of Functionality from Untrusted Control Sphere
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function

SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards

También se aplicarán los controles definidos en el **SCSVS** (Smart Contract Security Verification Standard), que sirven de guía para identificar posibles vulnerabilidades y medir el estado de seguridad y madurez de los Smart Contracts. Así mismo, se han tomado conceptos del **DASP top 10** (Decentralized Application Security Project), el cual define los 10 tipos más comunes de vulnerabilidades presentes en los Smart Contracts.

Se han efectuado baterías de pruebas mediante herramientas automatizadas y procedimientos manuales, de cara a identificar posibles vulnerabilidades y validar las mismas.

Estado

Presentamos un resumen del estado de seguridad del Smart Contract proporcionado por el cliente, tras la evaluación de las diferentes vulnerabilidades y deficiencias encontradas, en lo referente al riesgo al que está expuesta:

Nivel de seguridad	ALTO

Vulnerabilidades encontradas

A continuación, se exponen las vulnerabilidades y deficiencias encontradas, así como sus posibles efectos. Todas ellas serán explicadas y detalladas posteriormente en el apartado correspondiente a su criticidad.

El siguiente gráfico muestra el resumen del número total de vulnerabilidades y deficiencias halladas según su criticidad.

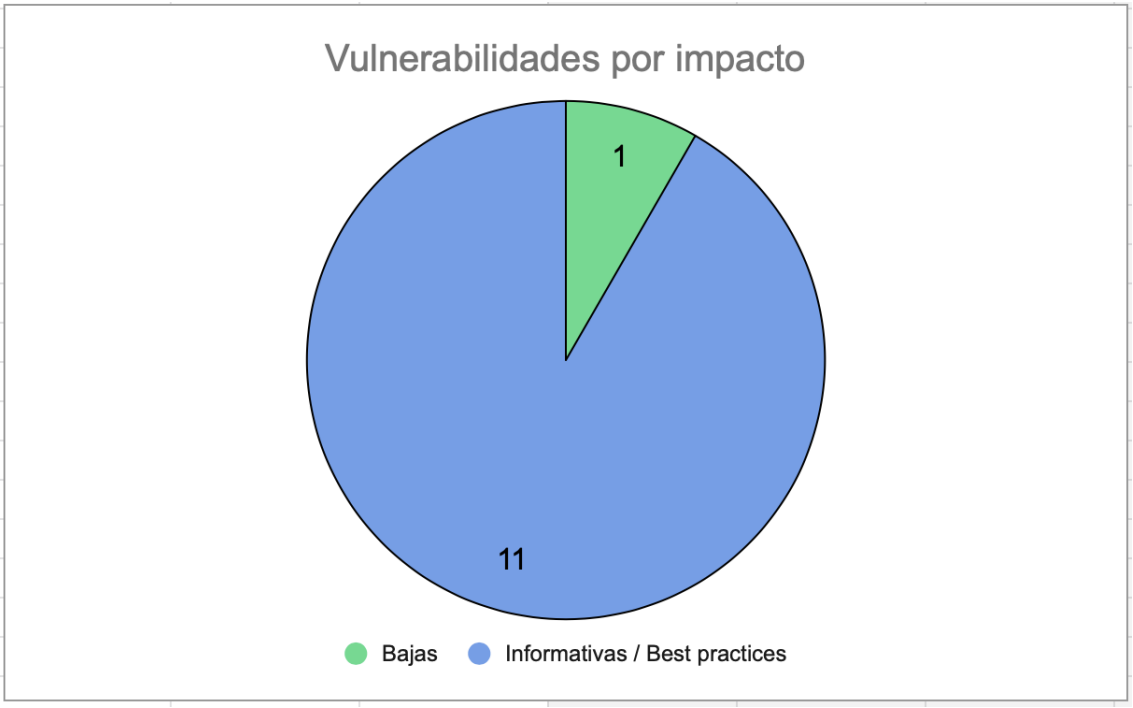

Fig. A. Vulnerabilidades según el impacto.

Resumen ejecutivo

El objeto del resumen ejecutivo es ofrecer una idea del trabajo realizado, de forma que pueda ser entendido o interpretado por personal no técnico o especializado en la materia.

Para entender el proceso de auditoría llevado a cabo, es necesario detallar brevemente cada paso que se ha seguido, no siendo necesario extenderse en ninguno de ellos, ya que ese es el objeto del informe técnico que se incluye en epígrafes posteriores.

El primer paso ha consistido en revisar el código reconociendo las especificaciones, fuentes y las instrucciones proporcionadas por el cliente con el objetivo de conocer el alcance, tamaño y la funcionalidad del Smart Contract a analizar.

Acto seguido, se ha realizado una revisión manual del código, consistente en analizar el código línea por línea en busca de posibles vulnerabilidades.

Posteriormente, mediante depuración manual y el uso de herramientas automatizadas, se ha realizado un análisis de la ejecución del código para validar los inputs que interaccionan con las distintas funciones del contrato.

También se ha hecho una revisión de buenas prácticas de desarrollo, proceso cuyo objetivo es analizar el Smart Contract para determinar mejoras en el rendimiento, eficiencia, eficacia, mantenibilidad, seguridad y control basado en prácticas académicas, investigaciones y recomendaciones.

A partir de los resultados anteriores, se ha procedido a buscar y detectar vulnerabilidades, tanto con pruebas manuales como con herramientas automatizadas.

A continuación, se enumeran las diferentes vulnerabilidades encontradas. En todas ellas se incluirá una descripción, su alcance, una propuesta de solución y la criticidad de los hallazgos clasificada según su impacto.

La métrica 'Impacto' define el grado de daño que se puede infligir a un Smart Contract desplegado en caso de que un atacante obtenga provecho de una vulnerabilidad en el mismo.

Informativo: Aunque la vulnerabilidad no presente un riesgo inmediato, debería de resolverse siempre que su comportamiento no sea el deseado por el cliente.

Bajo: El impacto de la vulnerabilidad no implica compromiso del Smart Contract; puede estar relacionado con fragmentos de código inutilizados o desactualizados.

Medio: El impacto de la vulnerabilidad no implica directamente el compromiso del Smart Contract, pero si puede poner en riesgo el resultado esperado a la hora de ejecutar un contrato en un determinado escenario.

Alto: El impacto de la vulnerabilidad supone compromiso del Smart Contract, provocando fallos en la ejecución, por ejemplo, acceso público a funciones críticas.

Crítico: El impacto de la vulnerabilidad supone compromiso del Smart Contract, ya que puede ser explotada y producir pérdida de tokens o manipulación de los datos.

Variable local y función con mismo nombre

IMPACTO BAJO

Descripción

La variable local con el nombre "owner" en la función CriptoVision.allowance() y CriptoVision._approve() tiene el mismo nombre que una función.

Elementos afectados:

- CriptoVision.allowance()
- CriptoVision._approve()

Impacto

Puede causar conflictos a la hora de llamar a la función o utilizar dicha variable.

Solución

Se recomienda renombrar las variables locales en las funciones CriptoVision.allowance() y CriptoVision._approve().

Referencias

https://swcregistry.io/docs/SWC-119

https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Uso de la función assembly

IMPACTO INFORMATIVO

Descripción

El uso de “assembly” es propenso a errores y se debe evitar su uso.

Elementos afectados:

- Address.isContract()
- Address._functionCallWithValue()

Impacto

El uso de dicha función puede provocar la proliferación de errores al ejecutar el código.

Solución

Se recomienda no usar la función “assembly”.

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

https://docs.soliditylang.org/en/v0.8.7/assembly.html

Uso incorrecto de variables booleanas

IMPACTO INFORMATIVO

Descripción

No es necesario comparar las variables booleanas con los valores “true” o “false” ya que es posible utilizar estas variables directamente.

Elementos afectados:

- Ownable.includeListed()
- Ownable.excludeListed()
- Ownable.isListed()
- Ownable.ContractLock()
- CriptoVision._approve()
- CriptoVision._transfer()

Impacto

Este tipo de prácticas añade complejidad al código, ya que son evitables.

Solución

Se recomienda eliminar estas comparaciones y utilizar las variables directamente.

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Funciones que no se utilizan

IMPACTO INFORMATIVO

Descripción

Se han encontrado funciones que no se emplean durante las pruebas de ejecución del código del Smart Contract.

Elementos afectados:

- Address._functionCallWithValue()
- Address.functionCall()
- Address.functionCall()
- Address.functionCallWithValue()
- Address.isContract()
- Address.sendValue()
- Context._msgData()
- SafeMath.div()
- SafeMath.mod()
- SafeMath.mul()
- SafeMath.sub()

Impacto

El uso de funciones que no son utilizadas añade complejidad al código, provocando que su interpretación y revisión sea un proceso más tedioso.

Solución

Se recomienda eliminar las funciones que no se utilizan y que han sido mencionadas previamente.

Referencias

https://swcregistry.io/docs/SWC-131
https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Expresión “this” redundante

IMPACTO INFORMATIVO

Descripción

La expresión “this” dentro de la función _msgData es redundante ya que no tiene ningún efecto.

Elementos afectados:

- Función _msgData dentro del contrato “Context”

Impacto

Esta expresión no genera valor ya que no realiza ninguna acción, por lo que no se modifica ni se crea código para dicha expresión.

Solución

Se recomienda eliminar la expresión “this” ya que añade complejidad al código y no aporta valor.

Referencias

https://docs.soliditylang.org/en/v0.8.9/internals/optimizer.html
https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Literal con demasiados dígitos

IMPACTO INFORMATIVO

Descripción

La variable _supply del contrato CriptoVision utiliza un literal con demasiados dígitos.

Elementos afectados:

- Variable _supply del contrato CriptoVision

Impacto

El uso de literales con demasiados dígitos dificulta la lectura y revisión.

Solución

Se recomienda utilizar el sufijo Ether cuyo efecto es equivalente a una multiplicación por una potencia de 10 o, en su defecto, usar la notación científica.

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Versión de Solc no fijada

IMPACTO INFORMATIVO

Descripción

Existen numerosas versiones del compilador de línea de comandos de Solidity, "solc", las cuales son lanzadas en periodos de tiempo bastante cortos.

Se ha verificado desde el propio código del Smart Contract que se está empleando una versión de pragma **^0.8.5**, lo que indica que puede funcionar hasta la versión 0.9.0.

Elementos afectados:

- pragma solidity ^0.8.5;

Impacto

Al no definir una versión fija de pragma, puede provocar que el bytecode producido varíe entre las diferentes compilaciones, dando lugar a errores o fallos de implementación. Esto es especialmente importante en el caso en que se realice la verificación del código a nivel de bytecode.

Solución

Se recomienda usar la misma versión para todas las compilaciones.

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable y modificador inutilizados

IMPACTO INFORMATIVO

Descripción

Se ha identificado la variable _previousOwner dentro del contrato "Ownable" pero no es usada en ningún momento durante su ejecución. Del mismo modo, el modificador "isListed" del contrato "Ownable" tampoco es usado.

Elementos afectados:

- Variable _previousOwner en el contrato "Ownable"
- Modificador isListed en el contrato "Ownable"

Impacto

Declarar variables o modificadores que nunca son empleados durante la ejecución del código no aporta valor. Dificulta la revisión y lectura del código a la hora de depurarlo, **además de requerir mayor cantidad de gas en el despliegue**.

Solución

Se recomienda eliminar dicha variable y modificador..

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
https://swcregistry.io/docs/SWC-131

Variables que deben ser declaradas como constantes

IMPACTO INFORMATIVO

Descripción

Se han encontrado variables que no se modifican durante la ejecución del Smart Contract.

Elementos afectados:

- Variable "_symbol" en el contrato de CriptoVision
- Variable "_decimals" en el contrato de CriptoVision
- Variable "_name" en el contrato de CriptoVision
- Variable "_previousOwner" en el contrato Ownable

Impacto

Para ejecutarse, los Smart Contracts requieren gas, que se utiliza como medida del coste computacional de cualquier transacción. Por ello, si las variables no se declaran como constantes en los casos que no modifican su valor, la ejecución del programa consumirá mayor cantidad de gas.

Solución

Se recomienda añadir el atributo "constant" a las variables que no modifican su valor en el transcurso de la ejecución del código.

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
https://www.binance.com/es/blog/all/todo-lo-que-necesitas-saber-sobre-las-criptomonedas-4214998246849022O1

Función pública que debe declararse como external

IMPACTO INFORMATIVO

Descripción

Se ha verificado el uso de funciones declaradas como “public” que nunca son llamadas por el propio contrato.

Elementos afectados:

- checkBlackList(address)
- owner()
- name()
- symbol()
- decimals()
- totalSupply()
- balanceOf(address)
- transfer(address,uint256)
- allowance(address,address)
- approve(address,uint256)
- transferFrom(address,address,uint256)

Impacto

Para ejecutarse, los Smart Contracts requieren gas, que se utiliza como medida del coste computacional de cualquier transacción. Por ello, si las funciones no se declaran como “external” en los casos que no son llamadas por el contrato, la ejecución del programa consumirá mayor cantidad de gas.

Solución

Se recomienda añadir el atributo “external” a dichas funciones.

Referencias

https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

INFORME TÉCNICO

Introducción

Bajo este epígrafe se engloban aquellas pruebas que se realizarán sobre el Smart Contract objeto de auditoría, con la finalidad de estudiar su estructura, características e implementación, de cara a encontrar vulnerabilidades y/o fallos de optimización.

Revisión de código

Como paso preliminar, con el objetivo de comprender la funcionalidad completa del Smart Contract, se ha revisado manualmente el código del fichero "CriptoVision.sol". En éste hay descritas dos librerías, una interfaz, dos contratos abstractos y un contrato propiamente dicho, los cuales se describen a continuación:

- Interfaz BEP20

Interfaz estándar para la creación de tokens en la Binance Smart Chain.

- Librería SafeMath

Librería estándar para el control de operaciones aritméticas y prevenir de desbordamientos de memoria en enteros.

- Librería Address

Librería estándar para la interacción con direcciones (wallets) en los Smart Contracts.

- Contrato abstracto Context

Contrato cuya finalidad es la de proporcionar información sobre el contexto de la ejecución actual. En él se incluyen funciones que proporcionan los datos relativos al remitente de la transacción y los datos de la transacción en sí

- Contrato abstracto Ownable

Hereda de **Context**.

Contrato en el que se definen parámetros para guardar la dirección del propietario actual, la dirección del propietario anterior, un bloqueo para que solo el propietario pueda interactuar con el contrato durante un período de

tiempo y una lista para las direcciones que pueden ser utilizadas. Además de estos parámetros, se definen tres modificadores.

- Contrato CriptoVision

Hereda de **Context** y **Ownable** e implementa **BEP20**.

Utiliza **Safemath** y **Address**.

Define los siguientes parámetros:

- private mapping (address => uint256) _balancesOf
- private mapping (address => mapping (address => uint256)) _allowances
- private string _name inicializado con el valor “CRIPTOVISION”
- private string _symbol inicializado con el valor “VISION”
- private uint8 _decimals inicializado con el valor 8
- private uint256 _supply inicializado con el valor 1000000000 * 10 ** _decimals

Define las siguientes funciones:

- constructor - introduce todos los tokens de CriptoVision en la dirección desde la que se hace el despliegue y emite un evento de transferencia.
- name - función pública que retorna el nombre del token.
- symbol - función pública que retorna el valor de la variable _symbol.
- decimals - función pública que retorna el valor de la variable _decimals.
- totalSupply - función pública que sobreescribe la función de la interfaz BEP20 y retorna la cantidad de tokens que siguen estando accesibles.
- balanceOf - función pública que sobrescribe una función de la interfaz BEP20 y retorna el balance de una cuenta.
- transfer - función pública que sobrescribe una función de la interfaz BEP20, llama a la función _transfer y retorna siempre el valor booleano “true”.
- allowance - función pública que sobrescribe una función de la interfaz BEP20 y retorna el número de tokens que tiene permitido gastar una cuenta (address) en nombre del propietario del contrato.
- approve - función pública que sobrescribe una función de la interfaz BEP20, llama a la función _approve y retorna siempre el valor booleano “true”.
- transferFrom - función pública que sobrescribe una función de la interfaz BEP20, que llama a la función _transfer y a la función _approve y emite un evento de tipo Transfer.
- _approve - función privada con bloqueo que aprueba la transferencia de tokens a una dirección específica y emite un evento de Approval.

- _transfer - función interna, virtual y con bloqueo que transfiere tokens de una cuenta a otra especificada.

Proceso de análisis

Tras la revisión del código completo de forma manual, con el objetivo de comprender el funcionamiento del mismo, se ha procedido a hacer un análisis estático para detectar las vulnerabilidades más comunes de los Smart Contracts escritos en Solidity.

Para ello, se ha empelado una serie de herramientas automatizadas. Inicialmente, se ha lanzado **Slither**, la cual abarca tanto vulnerabilidades como problemas de calidad del código.

Num	Check	What it Detects	Impact	Confidence
1	abiencoderv2-array	Storage abiencoderv2 array	High	High
2	array-by-reference	Modifying storage array by value	High	High
3	incorrect-shift	The order of parameters in a shift instruction is incorrect.	High	High
4	multiple-constructors	Multiple constructor schemes	High	High
5	name-reused	Contract's name reused	High	High
6	public-mappings-nested	Public mappings with nested variables	High	High
7	rtlo	Right-To-Left-Override control character is used	High	High
8	shadowing-state	State variables shadowing	High	High
9	suicidal	Functions allowing anyone to destruct the contract	High	High
10	uninitialized-state	Uninitialized state variables	High	High
11	uninitialized-storage	Uninitialized storage variables	High	High
12	unprotected-upgrade	Unprotected upgradeable contract	High	High
13	arbitrary-send	Functions that send Ether to arbitrary destinations	High	Medium
14	controlled-array-length	Tainted array length assignment	High	Medium
15	controlled-delegatecall	Controlled delegatecall destination	High	Medium
16	reentrancy-eth	Reentrancy vulnerabilities (theft of ethers)	High	Medium
17	storage-array	Signed storage integer array compiler bug	High	Medium
18	unchecked-transfer	Unchecked tokens transfer	High	Medium
19	weak-prng	Weak PRNG	High	Medium
20	enum-conversion	Detect dangerous enum conversion	Medium	High
21	erc20-interface	Incorrect ERC20 interfaces	Medium	High
22	erc721-interface	Incorrect ERC721 interfaces	Medium	High
23	incorrect-equality	Dangerous strict equalities	Medium	High
24	locked-ether	Contracts that lock ether	Medium	High
25	mapping-deletion	Deletion on mapping containing a structure	Medium	High
26	shadowing-abstract	State variables shadowing from abstract contracts	Medium	High
27	tautology	Tautology or contradiction	Medium	High
28	write-after-write	Unused write	Medium	High

Ilustración 1 - Lista de detectores de vulnerabilidades de Slither

```
CriptoVision.allowance(address,address).owner (CriptoVision.sol#521) shadows:
        - Ownable.owner() (CriptoVision.sol#472-474) (function)
CriptoVision._approve(address,address,uint256).owner (CriptoVision.sol#550) shadows:
        - Ownable.owner() (CriptoVision.sol#472-474) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Address.isContract(address) (CriptoVision.sol#265-276) uses assembly
        - INLINE ASM (CriptoVision.sol#272-274)
Address._functionCallWithValue(address,bytes,uint256,string) (CriptoVision.sol#391-419) uses assembly
        - INLINE ASM (CriptoVision.sol#411-414)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Ownable.includeListed(address) (CriptoVision.sol#457-460) compares to a boolean constant:
        -require(bool,string)(_list[account] == false,Account is already in list) (CriptoVision.sol#458)
Ownable.excludeListed(address) (CriptoVision.sol#462-465) compares to a boolean constant:
        -require(bool,string)(_list[account] == true,Account is already excluded from the list) (CriptoVision.sol#463)
Ownable.isListed() (CriptoVision.sol#432-435) compares to a boolean constant:
        -require(bool,string)(_list[_msgSender()] == false,Transaction Error) (CriptoVision.sol#433)
Ownable.ContractLock() (CriptoVision.sol#442-445) compares to a boolean constant:
        -require(bool,string)(_lock == false,Transaction Blocked) (CriptoVision.sol#443)
CriptoVision._approve(address,address,uint256) (CriptoVision.sol#549-560) compares to a boolean constant:
        -require(bool,string)(_list[spender] == false,Transaction Error) (CriptoVision.sol#557)
CriptoVision._approve(address,address,uint256) (CriptoVision.sol#549-560) compares to a boolean constant:
        -require(bool,string)(_list[owner] == false,Transaction Error) (CriptoVision.sol#556)
CriptoVision._transfer(address,address,uint256) (CriptoVision.sol#563-576) compares to a boolean constant:
        -require(bool,string)(_list[sender] == false,Transaction Error) (CriptoVision.sol#570)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Address._functionCallWithValue(address,bytes,uint256,string) (CriptoVision.sol#391-419) is never used and should be removed
Address.functionCall(address,bytes) (CriptoVision.sol#326-331) is never used and should be removed
Address.functionCall(address,bytes,string) (CriptoVision.sol#339-345) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (CriptoVision.sol#358-370) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (CriptoVision.sol#378-389) is never used and should be removed
Address.isContract(address) (CriptoVision.sol#265-276) is never used and should be removed
Address.sendValue(address,uint256) (CriptoVision.sol#294-306) is never used and should be removed
Context._msgData() (CriptoVision.sol#10-13) is never used and should be removed
SafeMath.div(uint256,uint256) (CriptoVision.sol#183-185) is never used and should be removed
SafeMath.div(uint256,uint256,string) (CriptoVision.sol#198-209) is never used and should be removed
SafeMath.mod(uint256,uint256) (CriptoVision.sol#222-224) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (CriptoVision.sol#237-244) is never used and should be removed
SafeMath.mul(uint256,uint256) (CriptoVision.sol#158-170) is never used and should be removed
SafeMath.sub(uint256,uint256) (CriptoVision.sol#125-127) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

CriptoVision._supply (CriptoVision.sol#485) is set pre-construction with a non-constant function or state variable:
        - 1000000000 * 10 ** _decimals
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state

Pragma version^0.8.5 (CriptoVision.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.7 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (CriptoVision.sol#294-306):
        - (success) = recipient.call{value: amount}() (CriptoVision.sol#301)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (CriptoVision.sol#391-419):
        - (success,returndata) = target.call{value: weiValue}(data) (CriptoVision.sol#400-402)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Variable Ownable._list (CriptoVision.sol#426) is not in mixedCase
Modifier Ownable.ContractLock() (CriptoVision.sol#442-445) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (CriptoVision.sol#11)" inContext (CriptoVision.sol#5-14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

CriptoVision.slitherConstructorVariables() (CriptoVision.sol#477-577) uses literals with too many digits:
        - _supply = 1000000000 * 10 ** _decimals (CriptoVision.sol#485)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Ownable._previousOwner (CriptoVision.sol#424) is never used in CriptoVision (CriptoVision.sol#477-577)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

CriptoVision._decimals (CriptoVision.sol#484) should be constant
```

Ilustración 2 - Resultado del análisis estático con Slither

```
Compiled with solc
Number of lines: 577 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 15
Number of informational issues: 33
Number of low issues: 2
Number of medium issues: 0
Number of high issues: 0

ERCs: ERC20

+--------------+-------------+-------+--------------------+--------------+----------+
|     Name     | # functions | ERCS  |     ERC20 info     | Complex code | Features |
+--------------+-------------+-------+--------------------+--------------+----------+
|   SafeMath   |      8      |       |                    |      No      |          |
|   Address    |      7      |       |                    |      No      | Send ETH |
|              |             |       |                    |              | Assembly |
| CriptoVision |      28     | ERC20 |     No Minting     |      No      |          |
|              |             |       | Approve Race Cond. |              |          |
|              |             |       |                    |              |          |
+--------------+-------------+-------+--------------------+--------------+----------+
```

Ilustración 3 - Resumen de los resultados proporcionados por Slither

Con ello, se han detectado **2 vulnerabilidades de impacto bajo, 33 vulnerabilidades de tipo informativo y 15 cuestiones relacionadas con la optimización de código**.

De cara a verificar las mismas, se ha generado una gráfica de las llamadas entre funciones, lo cual permite agilizar el análisis manual. De este modo ha sido posible descartar falsos positivos y excluir aquellas vulnerabilidades que no aplican para el caso concreto del contrato analizado.

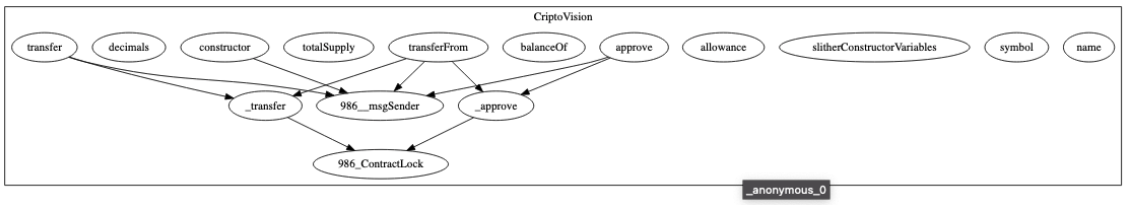

Ilustración 4 - Gráfica de llamadas a funciones

De cara a contrastar los resultados anteriores, se ha lanzado la herramienta **Mythx**, la cual no detecta ninguna vulnerabilidad ni fallo de implementación en el código del Smart Contract.

```
ArbitraryJump: Caller can redirect execution to arbitrary bytecode locations
ArbitraryStorage: Caller can write to arbitrary storage locations
ArbitraryDelegateCall: Delegatecall to a user-specified address
PredictableVariables: Control flow depends on a predictable environment variable
TxOrigin: Control flow depends on tx.origin
EtherThief: Any sender can withdraw ETH from the contract account
Exceptions: Assertion violation
ExternalCalls: External call to another contract
IntegerArithmetics: Integer overflow or underflow
MultipleSends: Multiple external calls in the same transaction
StateChangeAfterCall: State change after an external call
AccidentallyKillable: Contract can be accidentally killed by anyone
UncheckedRetval: Return value of an external call is not checked
UserAssertions: A user-defined assertion has been triggered
```

Ilustración 5 - Detectores de la herramienta Mythx

```
jlafuente@MacBook-Pro-de-Jorge CRIPTOVISION-NEW-TOKEN % docker run -v $(pwd):/tmp mythril:latest -v 1 analyze /tmp/CriptoVision.sol --solv 0.8.5
The analysis was completed successfully. No issues were detected.
```

Ilustración 6 - Resultados del análisis estático con Mythx

```
Contract Address:

0 functions are deemed safe in this contract:


Contract CriptoVision:

16 functions are deemed safe in this contract: checkBlackList(address), approve(address,uint256), balanceOf(address), includeListed(ad
dress), symbol(), unlock(), name(), totalSupply(), decimals(), lock(), transferFrom(address,address,uint256), excludeListed(address),
account_info_rotate_tine(uint256), transfer(address,uint256), owner(), allowance(address,address)


Contract SafeMath:

0 functions are deemed safe in this contract:
```

Ilustración 6 – Análisis estático de seguridad de las funciones son Mythx

Con el objetivo de analizar el comportamiento de las funciones presentes en el Smart Contract y verificar las entradas de las mismas, se ha realizado un proceso de fuzzing. Se ha configurado la herramienta **Echidna** para crear una cantidad ingente de transacciones y observar si el comportamiento del Smart Contract es el esperado.

```
Overwrite PropertiesCriptoVisionTransferable.sol
TestCriptoVisionTransferable.sol already exist and will not be overwritten
Overwrite echidna_config.yaml
################################################
Update the constructor in TestCriptoVisionTransferable.sol
To run Echidna:
        echidna-test CriptoVision.sol --contract TestCriptoVisionTransferable --config echidna_config.yaml
```

Ilustración 7 - Creación de los ficheros necesarios para iniciar el proceso de fuzzing

```
import "./PropertiesCriptoVisionTransferable.sol";
contract TestCriptoVisionTransferable is PropertiesCriptoVisionTransferable {
    constructor() public{

        crytic_owner = 0x00A329C0648769a73aFAc7F9381E08fB43DbeA50;
        crytic_user = 0x00a329C0648769a73AFaC7f9381E08Fb43dBeA60;
        crytic_attacker = 0x00a329C0648769a73afAC7F9381e08fb43DBEA70;

        //
        //
        // Update the following if totalSupply and balanceOf are external functions or state variables:

        initialTotalSupply = totalSupply();
        initialBalance_owner = balanceOf(crytic_owner);
        initialBalance_user = balanceOf(crytic_user);
        initialBalance_attacker = balanceOf(crytic_attacker);
    }
}
```

Ilustración 8 - Direcciones de prueba para realizar el proceso de fuzzing

```
Echidna 1.7.2
Tests found: 11
Seed: -2002582641744733784
Unique instructions: 2689
Unique codehashes: 1
Corpus size: 11
Tests
crytic_totalSupply_consistant_ERC20Properties: PASSED!
crytic_approve_overwrites: PASSED!
crytic_self_transferFrom_to_other_ERC20PropertiesTransferable: FAILED!
*no transactions made*
crytic_zero_always_empty_ERC20Properties: PASSED!
crytic_self_transfer_ERC20PropertiesTransferable: FAILED!
*no transactions made*
crytic_self_transferFrom_ERC20PropertiesTransferable: FAILED!
*no transactions made*
crytic_revert_transferFrom_to_zero_ERC20PropertiesTransferable: PASSED!
crytic_revert_transfer_to_user_ERC20PropertiesTransferable: PASSED!
crytic_revert_transfer_to_zero_ERC20PropertiesTransferable: PASSED!
crytic_transfer_to_other_ERC20PropertiesTransferable: PASSED!
crytic_less_than_total_ERC20Properties: PASSED!
Campaign complete, C-c or esc to exit
```

Ilustración 9 - Resultados del ejercicio de fuzzing

Cabe mencionar que en este caso **se han pasado todos los test** salvo los relacionados con el estándar ERC20, ya que al ir alojado en la Binance Smart Chain, el estándar que implementa el contrato es el BEP20 y no el ERC20.

Junto a ello, también se ha realizado un análisis estático manual de cara a identificar estructuras que pudiesen ser vulnerables, siguiendo los controles especificados en las metodologías de referencia indicadas al inicio del presente informe.

A continuación, se ha procedido a realizar un análisis dinámico del Smart Contract, para lo que ha sido necesario compilarlo y ejecutarlo.

Por medio de la herramienta **Manticore** se ha hecho un análisis simbólico del contrato, explorando todos los caminos de ejecución posibles. Sustituyendo las entradas por parámetros simbólicos ha permitido evaluar las condiciones que determinan la ejecución de cada bloque del contrato, no obteniéndose resultados positivos.

Ilustración 10 - Análisis dinámico con Manticore

Finalmente, se ha hecho un análisis manual del contrato empleando la herramienta **Remix** para compilar y depurar el Smart Contract.

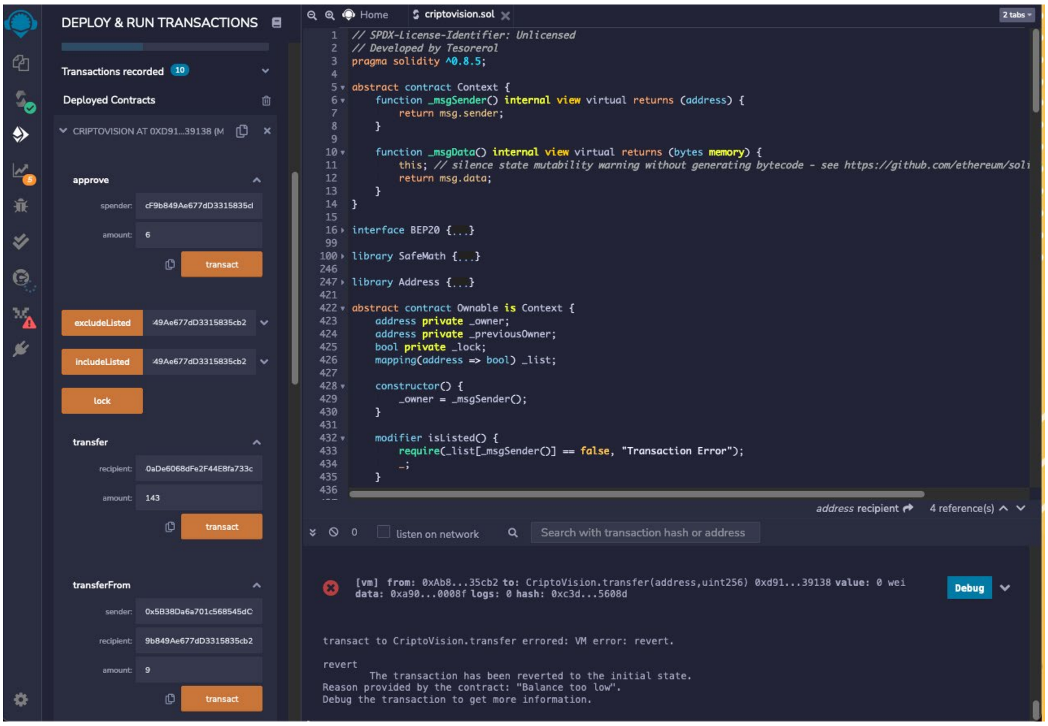

Ilustración 11 - Análisis dinámico manual con Remix

CONCLUSIONES

El Smart Contract dentro del alcance, empleado para interactuar con el token VSION de CRIPTOVISION, ha pasado por un proceso de auditoría de seguridad a través de procedimientos manuales en conjunción con herramientas automatizadas.

En base a las pruebas realizadas a lo largo de este proceso de auditoría, se puede concluir que **el estado de seguridad del Smart Contract es ALTO, siendo el nivel de riesgo al que está sometido BAJO**.

Es recomendable resolver las vulnerabilidades de bajo impacto detectadas y, en la medida de lo posible, subsanar los problemas de carácter informativo de cara a la optimización del contrato.

Cabe apuntar que el código objeto de la presente auditoría no ha sido comentado en su totalidad. Si bien se aprecian comentarios en la parte del contrato referente a librerías e interfaces estándar, es recomendable que todas las funciones, en especial las públicas sean comentadas, tal y como se recomienda en la guía de estilo del lenguaje de programación solidity. De esta forma se pretende que el código sea más sencillo de leer y entender por parte de otros desarrolladores, de cara a añadir nuevas funcionalidades o subsanar fallos.

Recordamos que una vez que el contrato sea desplegado en una dirección específica no podrá ser modificado o eliminado, lo que provoca que la presencia de vulnerabilidades deba ser subsanada previamente al despliegue. Por ello, se recomienda disponer de test unitarios exhaustivos. La utilización de este tipo de test podría evitar que se produzcan fallos inesperados en los contratos, asegurándose que se despliegan con la máxima seguridad y calidad posible.