Auditoría XBetters smart contract

Ventura Lucena Martínez

9 de marzo de 2023

Resumen

El objetivo de este informe es documentar tanto potenciales fallos como mejoras sobre el contrato inteligente de XBetters, desplegado sobre la red de Ethereum en la dirección 0xD19056371236ED978aa2e23699bB5EfAd0Bc3566. Para ello, se usará el código fuente del archivo *XBetters.sol* alojado en la anterior dirección de Etherscan como sus respectivas librerías. En este sentido, se comenzará haciendo unos comentarios respecto al procedimiento seguido, la estructura y las librerías, continuando con un estudio del cuerpo del contrato y finalizando con unas conclusiones. Por último, se añaden unos anexos con información que pueda ser de utilidad.

Índice

Índice de figuras

1. Procedimiento

En el presente informe se han considerado las siguientes características:

1. Seguridad del código.
2. Buenas prácticas.

Para ello, se ha realizado:

- **Análisis manual**: se ha estudiado la estructura del código y se han buscando vulnerabilidades que puedan afectar de forma servera a la seguridad del mismo.
- **Análisis automático**: se han utilizado herramientas que ayuden a examinar los casos de uso de las funciones del código de manera automatizada.

Se han categorizado las potenciales incidencias de la siguiente manera:

- **Incidencias de nivel alto**: problemas críticos que ponen en grave peligro la integridad y seguridad del proyecto teniendo consecuencias desastrosas como la pérdida de fondos, entre otras. Los contratos no deben ponerse en marcha antes de que se solucionen estos problemas.
- **Incidencias de nivel medio**: influyen en el funcionamiento actual del proyecto. Se incluyen dentro de esta categoría los bugs o la pérdida de de ingresos potenciales, así como posibles problemas relacionados con una gestión incorrecta del sistema. Es muy recomendable abordarlos.
- **Incidencias de nivel bajo**: no afectan directamente al proyecto. Problemas que puedan crear conflicto en futuras versiones del código o mejoras de rendimiento se introducirán en esta categoría.

2. Análisis

2.1. Estructura

Se recomienda hacer uso de la estructura de código disponible en la guía de estilo de Solidity [1], la cual permite identificar de manera eficiente las funciones según la visibilidad de estas. Por otro lado, también se sugiere hacer uso del estándar de comentarios NatSpec [2], que enriquece la documentación de código, haciendo más eficiente y sencillo su análisis.

2.2. Librerías importadas

Se analizarán de manera exhaustiva el conjunto de librerías que no correspondan a una importación desde un repositorio externo, como la del estándar ERC721A y las del directorio “opensea”, la cual evidencia una potencial modificación en el código fuente respecto al original [3] debido a que la importación viene de un fichero local. En cuanto al resto, se reflejarán si han sufrido cambios respecto a su alojamiento original respecto a los originales de los repositorios de OpenZeppelin.

2.2.1. ERC721A.sol

Para comprobar si existen cambios en el fichero se ha descargado una versión del repositorio con fecha 10 de noviembre de 2022 [4], días antes[1] del despliegue del contrato XBetters en la red de Ethereum bajo la transacción 0xacfd439301780ab8387a93628cb4d10b2045295ce5b9e0c35a6a985016242a0f a día 16 de noviembre de 2022. Se han compilado ambos (el original y el obtenido del directorio del contrato de Xbetters) y se han comparado los *bytecodes*. Al ser distintos, se ha comparado manualmente, detectándose los siguientes cambios:

- La función `approve()` ha sido modificada: se han refactorizado las funciones `_approve internal virtual` (líneas 889 y 906) en una única, eliminando el parámetro de tipo booleano `approvalCheck`. Puede encontrarse el respectivo test en [5, 6]. Ejecución sin fallos.

2.2.2. Ownable.sol

Sin cambios.

2.2.3. MerkleProof.sol

Sin cambios.

2.2.4. Strings.sol

Sin cambios. La librería String.sol importa la librería Math.sol de OpenZeppelin, aunque únicamente hace uso de dos funciones: `log10()` y `log256()`. Eliminar el resto del contenido que no se usa beneficia en:

- Ahorro de espacio en la cadena de bloques. El *bytecode* de las funciones no utilizadas se está almacenando en la cadena de bloques de manera innecesaria, además de aumentar el tamaño global del contrato y hacer que su despliegue sea más costoso.
- Código más legible.
- Menos código "atacable".

2.2.5. Operator-Filter-Registry

Nuevamente, se han comprobado la diferencia entre ficheros del directorio "opensea" con los originales a fecha de 9 de noviembre de 2022 [7]. Sin cambios. Conviene revisar los cambios de cara a las siguientes implementaciones según la auditoría de OpenZeppelin del 20 de enero de 2023 [8].

2.3. XBetters.sol

A continuación se hacen las siguientes observaciones respecto al contrato original de XBetters[2].

2.3.1. Variables estado o *state variables*

Constantes y no constantes En las líneas 37, 38 y 39 del contrato [9] se declaran 3 variables privadas con la nomenclatura de variables constantes según la guía de estilo oficial de Solidity [10]. Se recomienda que, al no tener intencionalidad de variables constantes, declararlas con una estructura *camelCase* [11].

[1] Se ha dejado un margen entre fechas debido a que es probable que el desarrollo haya acabado bastante antes que la fecha de despliegue.

[2] Únicamente se mencionarán propuestas de mejora o errores. La ausencia de comentarios respecto a alguna parte del contrato implica que no se proponen cambios.

Inicialización En Solidity, las variables que no son inicializadas con ningún valor quedan inicializadas con valores por defecto que el lenguaje establece. En este sentido, algunos ejemplos son:

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.17;
3
4  contract Default {
5      bool public a; // False
6      int public b; // 0
7      uint256 public c; // 0
8      address public d; // 0x0000000000000000000000000000000000000000
9      bytes32 public e; //
   ↪  0x0000000000000000000000000000000000000000000000000000000000000000
10     mapping(uint256 => uint256) public f; // Default value for introduced key.
11     string public g; // ""
12     // Note: Fixed arrays, not dynamic ones.
13     uint256[3] public h; // [0, 0, 0]
14
15     enum Phase {
16         Before,
17         WhitelistA,
18         WhitelistB,
19         Public,
20         Soldout,
21         Reveal
22     }
23
24     Phase public phase; // 0
25 }
```

En la línea 45 [12] del contrato se puede observar la variable `revealed` de tipo *bool* inicializada a *false*, y en la línea 60 [13] la variable `phase` del tipo *enum* `Phase` queda inicializada a `Phase.Before`. A la hora de desplegar el contrato, esto supone un incremento innecesario de +2267 unidades de gas sobre la inicialización la variable `revealed` y +2299 unidades de gas sobre la variable `phase`. Inicializar las variables sin asignarle ningún valor supone un ahorro del (-5.56 %) y (-5.56 %) en unidades de gas, respectivamente.

2.3.2. *setPhase()*

El contrato dispone de una variable de tipo *enum* llamada `Phase`, cuya finalidad es almacenar el estado actual del contrato respecto al ERC721A. Este tipo de variables permiten definir un conjunto de constantes con un nombre, a las cuales se les asocia un valor de tipo entero sin signo `uint256` de forma predeterminada, que va desde 0 hasta $2^{256} - 1$ [14]:

Enum constant	Valor
Before	0
WhistelistA	1
WhistelistB	2
Public	3
Soldout	4
Reveal	5

A su vez, el contrato dispone de un *setter* para este *enum*, el cual modifica el estado de la variable. Este *setter* recibe como parámetro un entero con signo `int` que posteriormente es casteado al tipo `Phase` para establecer finalmente el valor final de la fase. La función debería de recibir un entero sin signo `uint256`, tal y como espera una enumeración ya que, de esta manera, la función acepta valores como -1, los cuales provocan una transacción fallida debido a un `Panic error` con código 0x21 [15]. Este tipo de errores deben ser evitados en producción, tal y como se comenta en la anterior referencia:

> *"(...) Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix. Language analysis tools can evaluate your contract to identify the conditions and function calls which will cause a Panic."* [15]

Para evitar esto, sería conveniente realizar un control de errores, si bien se puede optimizar reestructurando la función de la siguiente manera:

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.17;
3
4  import "@openzeppelin/contracts/access/Ownable.sol";
5
6  contract PhaseTest is Ownable {
7      enum Phase {
8          Before,
9          WhitelistA,
10         WhitelistB,
11         Public,
12         Soldout,
13         Reveal
14     }
15     error InvalidPhase();
16
17     Phase public phase = Phase.Before;
18
19     // USING INT DATATYPE
20     // ------------------
21     // Original function
22     // Cost: 5899
23     function setPhase(int _phase) external onlyOwner {
24         // Error handling here does increment costs, due to
25         // int to uint undirectly convertion and then the
26         // neccesary checks.
```

```
27          phase = Phase(_phase);
28      }
29
30      // USING PHASE DATATYPE
31      // --------------------
32      // Cost: 5786
33      function setPhase(Phase _phase) external onlyOwner {
34          phase = _phase;
35      }
36
37      // USING UINT256 DATATYPE
38      // ----------------------
39      // Cost_1: 5856 without error handling
40      // Cost_2: 5892 with error handling using custom error and revert
41      // Deployment: 357671
42      function setPhase(uint256 _phase) external onlyOwner {
43          if(_phase > uint8(type(Phase).max))
44              revert InvalidPhase();
45
46          phase = Phase(_phase);
47      }
48
49      // Cost_3: 5889 using require
50      // Deployment: 380890
51      function setPhase(uint256 _phase) external onlyOwner {
52          require(_phase < uint8(type(Phase).max), "Invalid phase.");
53
54          phase = Phase(_phase);
55      }
56  }
```

- Se podría implementar un control de errores dentro de la **función original**. Sin embargo, al ser la más cara de las opciones con un coste de 5899 unidades de gas y, al tener que aumentar la lógica al no poder realizar una conversión directa del tipo de dato `int` a `uint8` (para comparar rango de valores aceptados), se prefiere alguna de las propuestas siguientes.
- **Usar como parámetro el tipo de dato del *enum*** es la opción más barata con un costo de 5786 unidades de gas (-1.8 %), aunque también tiene un inconveniente: recibir el tipo de dato del *enum* implica que el rango de valores ya va implícito, por lo que todavía podría recibir un valor como el comentado anteriormente (-1), causando un `Panic error` con código 0x21. Es por ello que se propone alguno de los dos últimos ejemplos como solución.
- **Usar como parámetro un entero sin signo** `uint256`: esta opción permite realizar un control de errores como máximo en dos líneas de código (más la declaración del error personalizado) haciendo los respectivos casteos de variables de forma directa. Se han contemplado el uso de 1) `revert` con errores personalizados y 2) el uso de `require`, ya que aunque parezca que funciona de igual manera, esto no es así. La diferencia reside en cuánto cuesta cada despliegue: el uso de errores personalizados en conjunción con `revert` es más barato que hacer uso de `require`. En

este caso, el ahorro en el despliegue es de 23219 unidades de gas (-6.09 %).

2.3.3. Modificadores

De manera similar a lo comentado al final de la sección 2.3.2, se ha comparado el coste de gas en el despliegue del contrato con distintas codificaciones de los modificadores: 1) `revert` con errores personalizados y 2) el uso de `require`.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

contract ModifiersTest {

    address private fiatMinter = 0x349560B18AF0aC8474dFa15221C5430A94A5E3C6;

    error CallerIsOtherContract();
    error CallerIsNotMinter();

    // USING REQUIRE
    // -------------
    // Cost: 142132
    modifier callerIsUser() {
        require(tx.origin == msg.sender, "Caller is another contract");
        _;
    }

    modifier onlyFiatMinter() {
        require(fiatMinter == msg.sender, "Caller is not minter");
        _;
    }

    // USING REVERT WITH CUSTOM ERROR
    // Cost: 91884
    modifier callerIsUSer() {
        if (tx.origin != msg.sender) {
            revert CallerIsOtherContract();
        }
        _;
    }

    modifier onlyFiatMinter() {
        if (fiatMinter != msg.sender) {
            revert CallerIsNotMinter();
        }
        _;
    }

```

```
    // Cost using modifier with require:    2322
    // Cost using modifier with rever:      2322
    function foo() public view callerIsUSer onlyFiatMinter{}
}
```

El ahorro en el despliegue del contrato con los modificadores haciendo uso de condicionales y `revert` es de un (-35.35 %) sobre los modificadores con `require`.

Por otro lado, se ha comprobado que ambos modificadores funcionan correctamente, haciendo especial mención a `callerIsUser()`, el cual evita que se pueda llamar a alguna función de minteo a través de otro contrato (prevención contra bots).

2.3.4. Funciones de minteo

En esta sección se comentarán dos aspectos para cada función: 1) desempeño y 2) optimización.

whitelistAMint() y *whitelistBMint()*

- **Desempeño**: se ha comprobado el funcionamiento de la fase de minteo con *whitelist* con el objetivo de verificar el correcto desempeño de la gestión de la misma haciendo uso de un *Merkle Tree*. El resultado ha sido el esperado. Para ello, se ha utilizado el ejemplo disponible en el anexo B.
- **Optimización**: se propone, de igual manera que en anteriores secciones, el cambio de las sentencias `require` por condicionales con la sentencia `revert` con errores personalizados:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

contract WhitelistMintTest {

    enum Phase {
        Before,
        WhitelistA,
        WhitelistB,
        Public,
        Soldout,
        Reveal
    }

    (...)

    Phase public phase = Phase.Before;

    error CallerIsOtherContract();

    (...)

```

```
23      error WhitelistANotActive();
24      error NotWhitelistedInA();
25      error WhitelistBNotActive();
26      error NotWhitelistedInB();
27      error LimitExceeded(); // Shared with other mint functions.
28      error NotEnoughTokensLeft(); // Shared with other mint functions.
29      error UnsufficientEther(); // Shared with other mint functions.
30
31      // Cost with require: 4399403
32      // Cost with revert and custom error:
33      //      - Only with WL A: 4368770
34      //      - Both WL A and B: 4316494
35      function whitelistAMint(
36          uint _quantity,
37          bytes32[] calldata _proof
38      ) external payable callerIsUSer {
39          if (phase != Phase.WhitelistA)
40              revert WhitelistANotActive();
41
42          if (!isWhitelistedA(msg.sender, _proof))
43              revert NotWhitelistedInA();
44
45          if (_numberMinted(msg.sender) + _quantity > MAX_WLA_MINTS_PER_ADDRESS)
46              revert LimitExceeded();
47
48          // Code shared with other functions.
49          if (totalSupply() + _quantity > MAX_SUPPLY)
50              revert NotEnoughTokensLeft();
51
52          if (msg.value < (whitelistAMintPrice * _quantity))
53              revert UnsufficientEther();
54          // ------------------------------------
55
56          _safeMint(msg.sender, _quantity);
57      }
58
59      function whitelistBMint(
60          uint _quantity,
61          bytes32[] calldata _proof
62      ) external payable callerIsUSer {
63          if (phase != Phase.WhitelistB)
64              revert WhitelistBNotActive();
65
66          if (!isWhitelistedB(msg.sender, _proof))
67              revert NotWhitelistedInB();
68
```

```
69          if (_numberMinted(msg.sender) + _quantity > MAX_WLB_MINTS_PER_ADDRESS)
70              revert LimitExceeded();
71
72          // Code shared with other functions.
73          if (totalSupply() + _quantity > MAX_SUPPLY)
74              revert NotEnoughTokensLeft();
75
76          if (msg.value < (whitelistBMintPrice * _quantity))
77              revert UnsufficientEther();
78          // ---------------------------------------
79
80          _safeMint(msg.sender, _quantity);
81      }
82
83      (...)
84  }
```

No seguir esta metodología para la función *whitelistAMint()* implica un gasto adicional de 30633 unidades de gas, y no hacerlo con ambas funciones lo incrementaría 52276 unidades de gas más[3], ascendiendo la suma total a un gasto de 82909 unidades de gas.

publicMint()

- **Desempeño**: se ha comprobado el funcionamiento de la fase de minteo público para verificar su correcto desempeño. El resultado ha sido el esperado.
- **Optimización**: se propone, de igual manera que en anteriores secciones, el cambio de las sentencias `require` por condicionales con la sentencia `revert` con errores personalizados:

```
1   (...)
2
3   error PublicMintNotActive();
4
5   // Cost with require: 4316494
6   // Cost with revert and custom error: 4287661
7   function publicMint(uint _quantity) external payable callerIsUSer {
8
9       // Code shared with fiatMint().
10      if (phase != Phase.Public)
11          revert PublicMintNotActive();
12      if (_numberMinted(msg.sender) + _quantity > MAX_PUB_MINTS_PER_ADDRESS)
13          revert LimitExceeded();
14      // ---------------------------------------
15
16      // Code shared with other functions.
```

[3]Esto se debe a la reutilización de algunos errores personalizados para ambas funciones. Véanse líneas de código 27, 28 y 29 del apartado "Optimización" de la sección **??**.

```
    if (totalSupply() + _quantity > MAX_SUPPLY)
        revert NotEnoughTokensLeft();

    if (msg.value < (mintPrice * _quantity))
        revert UnsufficientEther();
    // ------------------------------------

    _safeMint(msg.sender, _quantity);
}

(...)
```

Esto supone un gasto adicional de 28833 unidades de gas.

fiatMint()

- **Desempeño**: se ha comprobado el funcionamiento de la fase de minteo FIAT para verificar su correcto desempeño. El resultado ha sido el esperado.
- **Optimización**: se propone, de igual manera que en anteriores secciones, el cambio de las sentencias `require` por condicionales con la sentencia `revert` con errores personalizados:

```
// Cost with require: 4287661
// Cost with revert and custom error: 4217369
function fiatMint(
    address _account,
    uint _quantity
) external onlyFiatMinter {

    // Code shared with PublicMint().
    if (phase != Phase.Public)
        revert PublicMintNotActive();
    if (_numberMinted(msg.sender) + _quantity > MAX_PUB_MINTS_PER_ADDRESS)
        revert LimitExceeded();
    // ------------------------------------

    // Code shared with other functions.
    if (totalSupply() + _quantity > MAX_SUPPLY)
        revert NotEnoughTokensLeft();
    // ------------------------------------

    _safeMint(_account, _quantity);
}
```

Esto supone un gasto adicional de 70292 unidades de gas.

Estas funciones pueden refactorizarse aún más haciendo uso de modificadores que alojen el código que compartan entre sí. En este sentido, se pueden crear un total de cuatro nuevos modificadores que acorten el código en cada función y que, dependiendo de cada uno, puedan ser más legibles. Estos modificadores se llamarán en cada función de minteo sin alterar el orden original:

```
modifier onlyIfThereAreTokensLeft(uint _quantity) {
    if (totalSupply() + _quantity > MAX_SUPPLY)
        revert NotEnoughTokensLeft();
    _;
}

modifier onlyIfEnoughEther(uint _quantity) {
    if (msg.value < (whitelistAMintPrice * _quantity))
        revert UnsufficientEther();
    _;
}

modifier onlyInPubliPhase() {
    if (phase != Phase.Public)
        revert PublicMintNotActive();
    _;
}

modifier checkMaxPublicMintPerAddress(uint _quantity) {
    if (_numberMinted(msg.sender) + _quantity > MAX_PUB_MINTS_PER_ADDRESS)
        revert LimitExceeded();
    _;
}
```

Aunque el despliegue de esta versión es más caro, con un incremento de 3609 unidades de gas.

2.3.5. *tokensOfOwner()*

En el bucle *for* se hace una llamada al método `_startTokenId()` del fichero ERC721A.sol. Mide el índice de comienzo desde que se inicia el contrato ERC721A en el constructor, el cual no puede ser cambiado post-despliegue. Dado que esta función no se ha modificado y de forma predeterminada devuelve 0, la inicialización de la variable `i` dentro del bucle se puede ahorrar, tal y como se ha visto en la sección 2.3.1:

```
// Cost per execution with _startTokenId() when it returns 0 (not overriden):     13357
↪  (Cost only applies when called by a contract)
// Cost per execution without _startTokenId() when it returns 0 (not overriden):  13322
↪  (Cost only applies when called by a contract)
function tokensOfOwner(
    address owner
) external view returns (uint256[] memory) {
    unchecked {
```

```
        uint256 tokenIdsIdx;
        address currOwnershipAddr;
        uint256 tokenIdsLength = balanceOf(owner);
        uint256[] memory tokenIds = new uint256[](tokenIdsLength);
        TokenOwnership memory ownership;

        for (uint256 i; tokenIdsIdx != tokenIdsLength; ++i) { // _startTokenId() = 0 in
↪  ./ERC721A.sol
            ownership = _ownershipAt(i);
            if (ownership.burned) {
                continue;
            }
            if (ownership.addr != address(0)) {
                currOwnershipAddr = ownership.addr;
            }
            if (currOwnershipAddr == owner) {
                tokenIds[tokenIdsIdx++] = i;
            }
        }
        return tokenIds;
    }
}
```

Mencionar que, el ahorro en gas en despreciable y únicamente se aplica el coste cuando se llama a la función desde un contrato. Aún así, por cada llamada a la función se consumen 35 unidades de gas extra.

2.3.6. *tokenURI()*

Se recomienda la sustitución de las sentencias `require` por condicionales con la sentencia `revert` con errores personalizados:

```
(...)

// Cost with require:                    4230849
// Cost with revert and custom error:    4207416
error TokenDoesNotExist();

function tokenURI(
    uint256 tokenId
) public view override returns (string memory) {
    if (!_exists(tokenId)) {
        revert TokenDoesNotExist();
    }

    if (revealed) {
        return string(abi.encodePacked(baseURI, Strings.toString(tokenId), ".json"));
```

```
    } else {
        return notRevealedUri;
    }
}

(...)
```

No seguir esta recomendación implica un gasto adicional de 23433 unidades de gas en el despliegue del contrato.

2.3.7. *getPhaseMaxValue()*

Al igual que ocurre en la sección 2.3.2, que la función reciba como parámetro un entero con signo `int` permite introducir datos que pueden ocasionar un `Panic error` con código 0x21 en tiempo de ejecución. Por ello, se recomienda usar como parámetro un entero sin signo `uint256` para realizar un control de errores que permita revertir la transacción de forma segura:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "@openzeppelin/contracts/access/Ownable.sol";

contract GetPhaseMaxValueTest is Ownable {

    uint256 private MAX_WLA_MINTS_PER_ADDRESS = 1000;
    uint256 private MAX_WLB_MINTS_PER_ADDRESS = 1000;
    uint256 private MAX_PUB_MINTS_PER_ADDRESS = 100;

    enum Phase {
        Before,
        WhitelistA,
        WhitelistB,
        Public,
        Soldout,
        Reveal
    }
    error InvalidPhase();

    Phase public phase = Phase.Before;

    function getPhaseMaxValue(
        uint256 _phase
    ) external view onlyOwner returns (uint256) {
        // Error handling
        if(_phase > uint8(type(Phase).max))
            revert InvalidPhase();

```

```
        if (Phase(_phase) == Phase.WhitelistA) {
            return MAX_WLA_MINTS_PER_ADDRESS;
        } else if (Phase(_phase) == Phase.WhitelistB) {
            return MAX_WLB_MINTS_PER_ADDRESS;
        } else if (Phase(_phase) == Phase.Public) {
            return MAX_PUB_MINTS_PER_ADDRESS;
        }

        return 0;
    }
}
```

2.3.8. *setPhaseMaxValue()*

Al igual que ocurre en anteriores secciones, la función permite introducir datos que pueden ocasionar un `Panic error` con código 0x21 en tiempo de ejecución. Por ello, se recomienda usar como parámetro un entero sin signo `uint256` para realizar un control de errores que permita revertir la transacción de forma segura. Por otro lado, la función permite ejecutar una transacción con valores de fase que carecen de sentido en la lógica del contrato: 1) 0 ó "Before", 2) 4 ó "Soldout" y 3) 5 ó "Reveal". Aunque no se asignen valores para dichas fases (no tiene efecto ya queda bien asignado bajo condicionales dentro de la función), sí que permite ejecutarla en caso de descuido, lo que conllevaría a pagar 24634 unidades de gas de promedio.

```
(...)

error PhaseNotMintable();

function setPhaseMaxValue(uint256 _phase, uint256 _value) external onlyOwner {
    // Error handling.
    if(_phase > uint8(type(Phase).max))
        revert InvalidPhase();

    if (Phase(_phase) == Phase.WhitelistA) {
        MAX_WLA_MINTS_PER_ADDRESS = _value;
    } else if (Phase(_phase) == Phase.WhitelistB) {
        MAX_WLB_MINTS_PER_ADDRESS = _value;
    } else if (Phase(_phase) == Phase.Public) {
        MAX_PUB_MINTS_PER_ADDRESS = _value;
    } else {
        // Error handling.
        revert PhaseNotMintable();
    }
}

(...)
```

Por último, se pueden establecer valores para cada una de las fases mayores al circulante total

de tokens `MAX_SUPPLY`. Si bien esto no es un error, un descuido a la hora de establecer el valor puede resultar en valores no deseados para una fase concreta, quedando el resto de fases sin tokens disponibles en el peor de los casos.

3. Resultados

Se han aplicado los cambios comentados anteriormente al contrato [16], con el objetivo de comparar resultados de manera global, más allá de los ya comentados. Se han utilizado tanto la librería Hardhat de JavaScript como el editor Remix para los informes de gas. Los resultados son los siguientes:

Solc version: 0.8.17	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas		
Methods					
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
XBetters	approve	-	-	51083	1	-
XBetters	fiatMint	-	-	81885	1	-
XBetters	publicMint	-	-	83531	2	-
XBetters	reveal	-	-	50821	1	-
XBetters	safeTransferFrom	-	-	90731	2	-
XBetters	safeTransferFrom	-	-	91605	2	-
XBetters	setBaseURI	-	-	33704	1	-
XBetters	setFiatMinter	-	-	29263	1	-
XBetters	setMerkleRootA	-	-	46458	3	-
XBetters	setMintPrice	-	-	29014	1	-
XBetters	setNotRevealedUri	-	-	38164	2	-
XBetters	setPhase	26357	29169	28700	6	-
XBetters	setPhaseMaxValue	-	-	29467	1	-
XBetters	setWhitelistAMintPrice	-	-	29059	1	-
XBetters	transferFrom	85662	87556	86609	2	-
Deployments				**% of limit**		
XBetters		-	-	5377661	17.9 %	-

Figura 1: Informe de gas para XBetters (Hardhat).

Solc version: 0.8.17	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas		
Methods					
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**usd (avg)**
XBetters_IP	approve	-	-	51061	1	-
XBetters_IP	fiatMint	-	-	81884	1	-
XBetters_IP	publicMint	-	-	83531	2	-
XBetters_IP	reveal	-	-	50821	1	-
XBetters_IP	safeTransferFrom	-	-	90731	2	-
XBetters_IP	safeTransferFrom	-	-	91605	2	-
XBetters_IP	setBaseURI	-	-	33687	1	-
XBetters_IP	setFiatMinter	-	-	29263	1	-
XBetters_IP	setMerkleRootA	-	-	46480	3	-
XBetters_IP	setMintPrice	-	-	29036	1	-
XBetters_IP	setNotRevealedURI	-	-	38146	2	-
XBetters_IP	setPhase	26379	29191	28722	6	-
XBetters_IP	setPhaseMaxValue	-	-	29504	1	-
XBetters_IP	setWhitelistAMintPrice	-	-	29037	1	-
XBetters_IP	transferFrom	85662	87556	86609	2	-
Deployments					**% of limit**	
XBetters_IP		-	-	5178650	17.3 %	-

Figura 2: Informe de gas para XBetters_IP (Hardhat).

Figura 3: Informe de gas para XBetters (Remix).

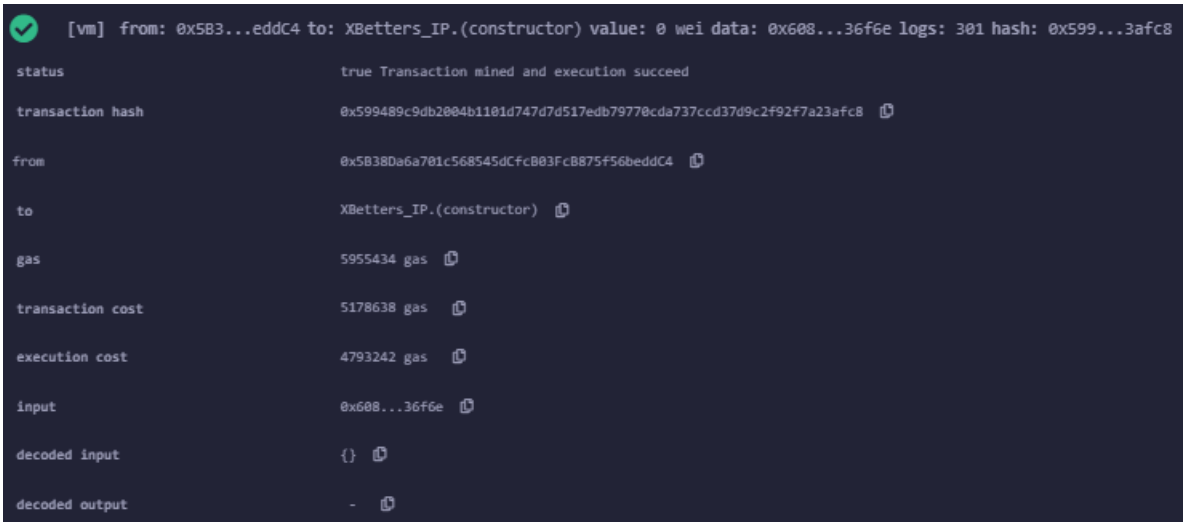

Figura 4: Informe de gas para XBetters_IP (Remix).

Graficando los datos, se observa como **la propuesta de mejora reduce los costes de gas**, siendo en **Gas** un -3.68 %, en el **Coste de Transacción** un 3.70 %, en el **Coste de Ejecución** un 3.70 % y en el **despliegue de hardhat** un 3.70 %:

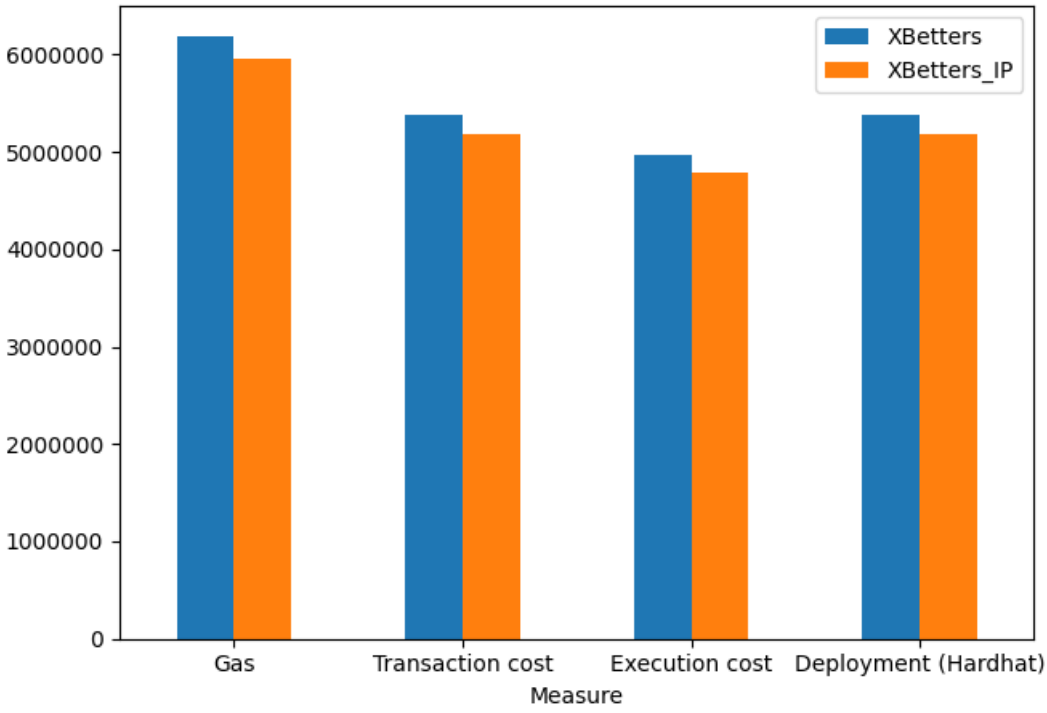

Figura 5: Gráfica de gas.

4. Conclusiones

En este informe se ha hecho una revisión del contrato inteligente XBetters. Aparentemente, no se han encontrado vulnerabilidades que puedan afectar de manera severa al contrato. Por otro lado, se han sugerido un conjunto de alternativas que mejoren, tanto la legibilidad y la estructura del contrato como el comportamiento actual del mismo, reduciendo el gasto en unidades de gas. Finalmente, se ha presentado una propuesta de mejora y se ha realizado un análisis global del gasto entre ambos contratos.

Incidencias de nivel alto No se han encontrado incidencias críticas.

Incidencias de nivel medio Si se introduce un valor no esperado en la siguientes funciones se devolverá un `Panic error` 0x21. Es recomendable su corrección.

- *setPhase()*: véase sección 2.3.2.
- *getPhaseMaxValue()*: véase sección 2.3.7.
- *setPhaseMaxValue()*: véase sección 2.3.8.

Incidencias de nivel bajo Se realizan propuestas de mejora de eficiencia en las siguientes funciones:

- String.sol: véase sección 2.2.4.
- Operator-Filter-Registry: véase sección 2.2.5.
- Variable estado o *state variables*: véase sección 2.3.1.
- Modificadores: véase sección 2.3.3.
- Funciones de minteo: véase sección 2.3.4.
- *tokensOfOwner*: véase sección 2.3.5.
- *tokenURI*: véase sección 2.3.6.

Referencias

[1] *Solidity style guide: Order of functions*. English. Ver. 0.8.17. Accedido el 7 de marzo de 2023. Ethereum Foundation. URL: https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-functions. (2021).

[2] *Solidity style guide: NatSpec Format*. English. Ver. 0.8.17. Accedido el 7 de marzo de 2023. Ethereum Foundation. URL: https://docs.soliditylang.org/en/v0.8.17/natspec-format.html. (2021).

[3] GitHub: Chiru-Labs. *Chiru-Labs/ERC721A*. Accedido el 3 de marzo de 2023. URL: https://github.com/chiru-labs/ERC721A.

[4] GitHub: Chiru-Labs. *Chiru-Labs/ERC721A (44ab010)*. Accedido el 6 de marzo de 2023. URL: https://github.com/chiru-labs/ERC721A/tree/44ab0103c2addf9e16bd164422b9a7359eac0a49.

[5] GitHub: VenturaLM. *VenturaLM/XBetters-audit/src/test/XBetters.js*. Accedido el 7 de marzo de 2023. URL: https://github.com/VenturaLM/XBetters-audit/blob/master/src/test/XBetters.js#L104.

[6] GitHub: VenturaLM. *VenturaLM/XBetters-audit/src/test/XBetters.js*. Accedido el 7 de marzo de 2023. URL: https://github.com/VenturaLM/XBetters-audit/blob/master/src/test/XBetters.js#L129.

[7] GitHub: ProjectOpenSea. *ProjectOpenSea/operator-filter-registry (419ce28)*. Accedido el 6 de marzo de 2023. URL: https://github.com/ProjectOpenSea/operator-filter-registry/tree/419ce2835a6f3bd5be4b14ac19278bb6060312fa.

[8] GitHub: ProjectOpenSea. *OpenSea Operator Filteer Audit Report*. Accedido el 6 de marzo de 2023. URL: https://github.com/ProjectOpenSea/operator-filter-registry/blob/main/audit/OpenSea\%20Operator\%20Filteer\%20Audit\%20Report.pdf.

[9] Ángel Moratilla. *Etherscan: XBetters.sol*. (2022). URL: https://etherscan.io/address/0xd19056371236ed978aa2e23699bb5efad0bc3566#code#F1#L37.

[10] *Solidity style guide: Constants*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: https://docs.soliditylang.org/en/v0.8.17/style-guide.html#constants. (2021).

[11] *Solidity style guide: Local and state variable names*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: https://docs.soliditylang.org/en/v0.8.17/style-guide.html#local-and-state-variable-names. (2021).

[12] Ángel Moratilla. *Etherscan: XBetters.sol*. (2022). URL: https://etherscan.io/address/0xd19056371236ed978aa2e23699bb5efad0bc3566#code#F1#L45.

[13] Ángel Moratilla. *Etherscan: XBetters.sol*. (2022). URL: https://etherscan.io/address/0xd19056371236ed978aa2e23699bb5efad0bc3566#code#F1#L60.

[14] *Solidity style guide: Enums*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: https://docs.soliditylang.org/en/v0.8.17/types.html#enums. (2021).

[15] *Solidity error handling: Panic via assert and Error via require*. English. Ver. 0.8.17. Accedido el 4 de marzo de 2023. Ethereum Foundation. URL: https://docs.soliditylang.org/en/v0.8.17/control-structures.html#panic-via-assert-and-error-via-require. (2021).

[16] GitHub: VenturaLM. *VenturaLM/XBetters-audit/src/contracts/XBetters_IP.sol*. Accedido el 7 de marzo de 2023. URL: https://github.com/VenturaLM/XBetters-audit/blob/master/src/contracts/XBetters_IP.sol.

[17] GitHub: VenturaLM. *VenturaLM/XBetters-audit/src/test/XBetters.js*. Accedido el 6 de marzo de 2023. URL: https://github.com/VenturaLM/XBetters-audit/blob/master/src/test/XBetters.js#L152.

A. Errores en los tests ajenos al contrato

Durante la realización de los tests se ha encontrado 1 error ajeno a la lógica del contrato:

1. `whitelistAMint()`: aunque la transacción se ejecute con una clave privada asociada a una dirección incluída en el *Merkle root*, hardhat revierte la transacción con el error `Not whitelisted`. Se ha comprobado en el editor Remix y la ejecución se ha dado sin defectos.

```
2) XBetters
     whitelistAMint(). This function tests a Merkle Proof
       FIXME: whitelistAMint(
                1,
                [
                  0x1ebaa930b8e9130423c183bf38b0564b0103180b7dad301013b18e59880541ae,
                  0x343750465941b29921f50a28e0e43050e5e1c2611a3ea8d7fe1001090d5e1436
                ]
              )

   AssertionError: Expected transaction NOT to be reverted with reason 'Not
 ↪  whitelisted', but it was
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
    at runNextTicks (node:internal/process/task_queues:65:3)
    at listOnTimeout (node:internal/timers:528:9)
    at processTimers (node:internal/timers:502:7)
    at Context.<anonymous> (test\XBetters.js:200:21)
```

B. Ejemplo usado en *whitelistAMint()* y *whitelistBMint()*

El ejemplo completo puede ser encontrado en [17].

```
WhitelistA Merkle Tree:
  - 0xb12e5b97c5c34aeb22d4e5f0061100c5072c240346e4d28a1a73659930fe90b2 (Root)
      - 0x343750465941b29921f50a28e0e43050e5e1c2611a3ea8d7fe1001090d5e1436 (Index 4)
          - 0x00314e565e0574cb412563df634608d76f5c59d9f817e85966100ec1d48005c0 (Index 0)
          - 0x8a3552d60a98e0ade765adddad0a2e420ca9b1eef5f326ba7ab860bb4ea72c94 (Index 1)
      - 0x8393e82ea28dfe71f8a1bfc8bfbe85da65aa4a7f2ceb7b2e356854fb5983c538 (Index 5)
          - 0xe9707d0e6171f728f7473c24cc0432a9b07eaaf1efed6a137a4a8c12c79552d9 (Index 2)
          - 0x5b1130ba602a5b64a86675b98193c8989dfff9db8956c9d2eac539828c115523 (Index 3)

---

En whitelistAMint()
  Verify:
    Address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
    Hash: 0xe9707d0e6171f728f7473c24cc0432a9b07eaaf1efed6a137a4a8c12c79552d9
```

```
  Merkle Root: 0xb12e5b97c5c34aeb22d4e5f0061100c5072c240346e4d28a1a73659930fe90b2
  Proof: ["0x1ebaa930b8e9130423c183bf38b0564b0103180b7dad301013b18e59880541ae",
↪  "0x343750465941b29921f50a28e0e43050e5e1c2611a3ea8d7fe1001090d5e1436"]
```