# Postgrado en Ingeniería y Arquitectura Blockchain. Ethereum: Seguridad y Buenas prácticas (DApps).

Versión 1.0

# PROYECTO MODULO 6
# Informe de Auditoria

Federico Cambero Fenoy
IEBS Business School

Mayo de 2023

# Tabla de Contenidos

## 3. Disclaimer.

El contenido de este informe de auditoría se proporciona “tal cual”, sin declaraciones y garantías de cualquier tipo. El autor y su empleador renuncia a cualquier responsabilidad por daños derivados de, o en relación con, este informe de auditoría. Los derechos de autor de este informe permanecen en el autor.

Esta auditoría ha sido realizada por
Federico Cambero Fenoy

# 4. Introducción.

## 4.1 Propósito de este informe

He sido contratado por IEBS para realizar una auditoría de seguridad de los Smart Contract:

- Iebs_Faillapop_DAO.sol
- Iebs_Faillapop_shop.sol
- Iebs_Faillapop_vault.sol

Los objetivos de esta auditoría son los siguientes:

a. Determinar posibles vulnerabilidades que podrían ser aprovechadas por un atacante.
b. Determinar errores de los Smart contract, que podrían conducir a un comportamiento inesperado.
c. Determinar el correcto funcionamiento de los Smart Contract.
d. Analizar si se han aplicado las mejores prácticas durante el desarrollo.
e. Hacer recomendaciones para mejorar la seguridad y legibilidad del código.

Este informe representa un resumen de los hallazgos.

Al igual que con cualquier auditoría de código, existe un límite en el que se pueden encontrar vulnerabilidades y aún pueden existir rutas de ejecución inesperadas.

El autor de este informe no garantiza la completa cobertura (ver disclaimer).

## 4.2 Base de código enviado para la Auditoria.

El código de los Smart Contract y el informe final para esta auditoria está en el repositorio:

https://github.com/fenoy70/Blockchain/tree/main/Proyecto

Las versiones usadas de los Smart Contracts son:

Iebs_Faillapop_DAO.sol
f3dc25aedec0918e8f497927e10fc2441b7de26e

Iebs_Faillapop_shop.sol
c41ce04fb081dc8f2776f22e210cb311feb897ae

Iebs_Faillapop_vault.sol
8f3e829c0089c9945ae0a4e4a3ca4887d64294eb

El tipo es: Solidity

La plataforma usada es: Ethereum.

### 4.3 Metodología.

La auditoría se ha realizado en los siguientes pasos:

1. Obtener una comprensión del propósito previsto del código base al leer la documentación disponible.

2. Código fuente automatizado y análisis de dependencia.

3. Análisis manual línea por línea del código fuente para vulnerabilidades de seguridad y uso de directrices de mejores prácticas, que incluyen, pero no limitan a:
    a. Análisis de condición de carrera.
    b. Problemas de subdesbordamiento/desbordamiento.
    c. Vulnerabilidades de gestión de claves.

4. Elaboración de Informes.

## 5. Como leer este reporte.

| Nivel de riesgo | Descripción |
|---|---|
| CRÍTICO | Vulnerabilidades que normalmente tienen una explotación directa que causa pérdida de fondos o bloqueo de fondos, manipulación de los datos almacenados o denegación de servicio completa. |
| ALTO | Vulnerabilidades que causan un impacto significativo en el funcionamiento del contrato o denegación de servicio parcial. |
| MEDIO | Vulnerabilidades que no causan una pérdida directa de fondos o de datos pero sí afectan al funcionamiento correcto o esperado del contrato. |
| BAJO | Situaciones en las que no se siguen buenas prácticas de seguridad, fallos que no entrañan un riesgo directo o ineficiencias en la ejecución. |

El estado de un problema puede ser uno de los siguientes:

1. Pendiente.
2. Confirmado.
3. Resuelto

Hay que tener en cuenta que las auditorías son un paso importante para mejorar la seguridad de los Smart contracts y pueden encontrar muchos problemas. Sin embargo, la auditoría de bases de código complejas tiene sus límites y un riesgo restante presente (ver disclaimer). Los usuarios del sistema deben tener precaución. Con el fin de ayudar con la evaluación del riesgo restante, proporcionamos una medida de los siguientes indicadores clave:

1. Complejidad del código.
2. Legibilidad del código.
3. Nivel de documentación.
4. Cobertura de las pruebas.

Incluimos una tabla con estos criterios abajo.
Tenga en cuenta que la alta complejidad o la baja cobertura no equivalen necesariamente a un mayor riesgo, aunque ciertos errores se detectan más fácilmente en las pruebas unitarias que en una auditoria de seguridad y viceversa.

## 6. Resumen ejecutivo.

Los contratos analizados están afectados por serías vulnerabilidades y no se recomienda su uso hasta haber solucionado las diferentes incidencias.
Un usuario con exceso de poder, dado su role podría quedarse con todo el stake de todos los usuarios y eliminar todas las ventas realizadas y quedando afectado los contratos "*iebs_Faillapop_shop"* y el "*iebs_Falillapop_vault"* explotando las vulnerabilidades de exceso de centralización que les afectan.
Además, sería posible realizar una denegación de servicio completa al contrato *"iebs_Faillapop_shop"*, ya que un atacante podría poner en modo vacaciones todas las ventas del contrato.
Se debe destacar también que un atacante podría cerrar las votaciones siempre a su favor, dada la vulnerabilidad de la función "endDispute" del contrato *"iebs_Faillapop_DAO"*.

## 7. Resumen de resultados.

| Nº | Descripción | Nivel de riesgo | Estado | contrato |
|---|---|---|---|---|
| 1 | Modo vacaciones activado arbitrariamente. | **Critico** | Pendiente | iebs_Faillapop_shop |
| 2 | Visibilidad incorrecta. | **Critico** | Pendiente | iebs_Faillapop_vault |
| 3 | Autorización usando tx.origin. | **Alto** | Pendiente | iebs_Faillapop_DAO |
| 4 | Exceso de centralización. | **Alto** | Pendiente | iebs_Faillapop_shop |
| 5 | Variable no inicializada permite terminar todas las votaciones. | **Alto** | Pendiente | iebs_Faillapop_DAO |
| 6 | El ether adicional de una compra no es devuelto. | **Bajo** | Pendiente | iebs_Faillapop_shop |

## 8. Detalle de resultados.

### 1. Modo vacaciones activado arbitrariamente.

**Nivel de Riesgo:** **Critico**.

**Impacto:** Contrato iebs_Faillapop_shop.sol

Un atacante puede activar el modo vacaciones a través de la función *"setVacationMode"* para cualquier usuario que desee. De esta forma, puede poner en modo vacaciones todas las ventas del contrato, causando una denegación de servicio completa

**Zona afectada del código:**

```
iebs_Faillapop_shop.sol línea: 272
function setVacationMode(bool _vacationMode) external {
for (uint i = 0; i < offerIndex; i++) {
if (offered_items[i].seller == msg.sender) {
if (_vacationMode && offered_items[i].state == State.Selling)
{
offered_items[i].state = State.Vacation;
} else if (!_vacationMode && offered_items[i].state ==
State.Vacation) {
offered_items[i].state = State.Selling;
}
}
}
}
```

**Recomendación:** Cambiar la visibilidad de external a internal y lo que haría sería que solo se pueda llamar desde otra función de tipo external pero que esté controlada y que el vendedor sea quien solicite que se le ponga en modo Vacaciones y solo aquel que tenga un role de ADMIN_ROLE pueda ejecutarla siempre y cuando haya una solicitud de cambio de estado por parte del vendedor.

**Estado:** **Pendiente.**

**2. Visibilidad Incorrecta.**

**Nivel de Riesgo:** **Critico.**

**Impacto:** Contrato iebs_Faillapop_vault.sol

Un atacante puede acceder a la función *"distributeSlashing"* dado que su visibilidad es public en lugar de internal. De esta manera, podrá incrementar la variable max_claimable_amount todo lo que desee para después drenar el contrato si es un usuario powerseller.

**Zona afectada del código:**

iebs_Faillapop_vault.sol línea 210

```
function distributeSlashing(uint amount) internal {
totalSlashed += amount;
(, bytes memory data) =
nftContract.call(abi.encodeWithSignature("totalPowersellers()"));
uint totalPowersellers = abi.decode(data, (uint256));
uint newMax = totalSlashed / totalPowersellers;
max_claimable_amount = newMax;
   }
```

**Recomendación:** Puntualización sobre este caso.

La visibilidad es internal, con lo cual no es el caso expuesto y por otro lado la función *"distributeSlashing"* se llama desde la función *"doSlash"* y para ejecutarla hay que tener DAO_ROLE con lo cual no puede ser cualquier atacante ni tampoco llamando a la función directamente.

Dicho esto si fuera el caso expuesto tendría que ser cambiada a "Internal" y solo llamada desde la función *"doSlash"* donde se controle con un require que el address sea != address(0) y además en la función *"distributeSlashing"* después del .call se controle que la llamada ha ido correctamente y sino ha ido correctamente hacer un revert.

**Estado:** **Pendiente.**

### 3. Autorización usando tx.origin.

**Nivel de Riesgo:** **Alto.**

**Impacto:** Contrato iebs_Faillapop_DAO.sol

El contrato FP_DAO realiza un control de acceso basado en tx.origin, lo que permitiría a un usuario malicioso orquestar un ataque de phishing contra el administrador y poder realizar llamadas privilegiadas suplantando a esta figura.
El atacante desplegará un smart contract intermedio y cuando el administrador legítimo hiciese una llamada a ese contrato se lanzaría una llamada arbitraria al contrato vulnerable.
Aunque el msg.sender fuese el contrato malicioso, el tx.origin sí será la dirección del usuario privilegiado.

**Zona afectada del código:**

`iebs_Faillapop_DAO.sol` línea: 93

```
modifier onlyShop() {
require(
msg.sender == shop_addr,
"Unauthorized"
);
_;
}
```

**Recomendación:** **En este caso no se está usando tx.origin**, ahora bien dicho esto lo conveniente es no usar tx.origin, sino que hay que usar msg.sender o bien a un usuario / grupo de usuarios bien definidos.
Hay que testear que el código solamente permite el acceso a usuarios / grupos de usuarios designados.
Lo que haría es en el constructor igualar a “shop_addr” al msg.sender.

**Estado:** **Pendiente.**

## 4. Exceso de centralización.

**Nivel de Riesgo:** **Alto.**

**Impacto:** Contrato iebs_Faillapop_shop.sol

El contrato FP_Shop contiene una función privilegiada removeMaliciousSale que elimina una venta y añade al vendedor a la lista negra, por ende, causando el slashing completo de su stake en el contrato FP_Vault.
Esta acción está limitada al usuario administrador, que podrá tomar libremente esta decisión. Esto permitiría, por ejemplo, que la organización eliminase todas las ventas y se quedase con el stake de los todos los usuarios para repartirlo entre otros.

**Zona afectada del código:**

`iebs_Faillapop_shop.sol` línea 321

```
function removeMaliciousSale(uint itemId) external onlyRole(ADMIN_ROLE) {
require(offered_items[itemId].seller != address(0), "itemId does not
exist");
if (offered_items[itemId].state == State.Pending) {
reimburse(itemId);
} else if (offered_items[itemId].state == State.Disputed) {
reimburse(itemId);
closeDispute(itemId);
}
// Seller should NOT be paid
closeSale(itemId, false);
blacklist(offered_items[itemId].seller);
}
```

**Recomendación:** Lo suyo es que el contrato o las funciones con riesgo se manejen por una comunidad y no por una persona o dirección concreta. Habría que hacer cambios para que si se quiere realizar este tipo de acciones por una dirección concreta haya que aprobarlos por una comunidad. Y no que esté todo centralizado en una persona o en un Role. Es decir, todo debe depender de una multisig. También se puede establecer retardos o timelock para que les dé tiempo a los usuarios del contrato revertir la situación.

**Estado:** **Pendiente.**

**5. Variable no inicializada permite terminar todas las votaciones.**

**Nivel de Riesgo:** **Alto.**

**Impacto:** Contrato iebs_Faillapop_DAO.sol

El contrato FP_DAO no ha inicializado la variable quorum a DEFAULT_QUORUM, por lo tanto, esta tomará un valor por defecto de cero ya que es un uint256.
Un quorum de cero permitirá a cualquier usuario terminar la votación de una disputa de manera arbitraria a través de la función *"endDispute"*. Permitiendo a un atacante con suficiente poder de voto cerrar una votación a su favor nada más votar, justo después de crearla cuando no haya apenas votos en contra.

**Zona afectada del código:**

```
iebs_Faillapop_DAO.sol línea 63
///@notice Min number of people to pass a vote
uint256 quorum;
```

**Recomendación:** Habría que inicializar la variable quorum a la cantidad establecida, o bien cambiar en la función la variable quorum por la variable DEFAULT_QUORUM que si está inicializada.

**Estado:** **Pendiente.**

### 6. El ether adicional de una compra no es devuelto.

**Nivel de Riesgo:** Bajo.

**Impacto:** Contrato iebs_Faillapop_shop.sol

El contrato FP_Shop permite a los usuarios enviar más Ether en su llamada del estrictamente necesario. Este Ether no es devuelto de ninguna manera y se quedará sin ser usado en el contrato de manera indefinida.

**Zona afectada del código:**

iebs_Faillapop_shop.sol línea 138

```
function doBuy(uint itemId) external payable {
require(offered_items[itemId].seller != address(0), "itemId does not
exist");
require(offered_items[itemId].state == State.Selling, "Item cannot be
bought");
require(msg.value >= offered_items[itemId].price, "Incorrect amount of
Ether sent");
offered_items[itemId].buyer = msg.sender;
offered_items[itemId].state = State.Pending;
emit Buy(msg.sender, itemId);
}
```

**Recomendación:** Habría que cambiar la función *"doBuy"* para que controle que el pago de la compra sea por el precio del articulo y no por mayor o igual.
Y si esto se quiere dejar así de esta manera lo suyo es que, una vez realizada la compra, se haga la operación donde se calcule el ehter sobrante y se haga un transfer a la dirección / usuario que ha comprado el artículo.

**Estado:** Pendiente.