Ethereum: Seguridad y Buenas Prácticas (DApps)

-Sprint 4-

Proyecto Final -Informe de Auditoria-

Jaime Contreras
Master en Blockchain, Metaverso y NFT's
Diciembre 2023

Tabla de Contenido

1. Enfoque y Metodología

La metodología de auditoría de contratos inteligentes es esencial para garantizar la seguridad y el funcionamiento correcto en el mundo de las Criptomonedas y Blockchain.
Por tal razón se ha definido un proceso que comienza revisando la documentación para comprender el contrato y sus funciones. Luego, analiza el diseño y la estructura, identificando posibles amenazas. Herramientas automáticas como Slither ayudan a encontrar vulnerabilidades. Las pruebas de cobertura son cruciales para asegurar una amplia cobertura de código. La revisión manual profundiza en la detección de vulnerabilidades críticas. La documentación de hallazgos y sugerencias de soluciones asegura que el contrato cumpla con estándares de seguridad. Esta metodología es fundamental para proteger activos y usuarios en este espacio emergente. A continuación las fases del proceso.

Fase	Categoría	Descripción
 **Entendimiento Inicial**	**Documentación y Especificaciones**	• Revisar whitepaper, wiki, docs técnicas. • Entender propósito, funciones, actores, assets, y casos de uso del contrato
 **Arquitectura y Código Alto Nivel**	**Diseño y Estructura del Código**	• Revisar estructura, librerías, herencia, modificadores. • Mapear relaciones entre funciones/contratos. • Identificar vectores de entrada/salida de datos y cripto-activos
 **Evaluación de Riesgos**	**Threat Model**	• Enumerar amenazas comunes (reentrancy, overflow, etc). • Identificar amenazas particulares del contrato. • Priorizar amenazas críticas
 **Análisis Estático**	**Herramientas Automatizadas**	• Ejecutar Slither. • Revisar vulnerabilidades reportadas. • Evaluar bugs, code smells, y exploits potenciales. • Calificar hallazgos por probabilidad e impacto
 **Análisis Dinámico**	**Coverage de Tests**	• Ejecutar con solidity-coverage. • Analizar líneas/ramas de código sin coverage. • Desarrollar pruebas adicionales para casos críticos
 **Lógica de Negocio**	**Análisis de Funcionalidad**	• Revisar llamadas externas y manejo de cripto-activos. • Validar transferencias de fondos. • Verificar entradas/salidas de Ether/tokens
 **Revisión de Código**	**Revisión Manual**	• Verificar inicialización y uso de variables. • Detectar condiciones de carrera y reentrancy. • Confirmar validaciones de entradas y salidas
 **Reporte y Cierre**	**Documentación de Hallazgos**	• Documentar vulnerabilidades y exploits. • Proveer escenarios de reproceso. • Sugerir mitigaciones y soluciones cuando sea posible

I. Lista de Contactos

Nombre	Organización	Rol	Email	Teléfono
Juan Pérez	CryptoSecure Inc.	Auditor Jefe	juan.perez@email.com	+34 555 1234
María Gómez	Blockchain Innov.	Desarrolladora	maria.gomez@email.com	+34 555 5678
Carlos López	Tech Solutions	Analista de Riesgo	carlos.lopez@email.com	+34 555 9012
Ana Pozo	EtherTrust	Consultora	ana.pozo@email.com	+34 555 3456
Santiago Ruiz	SmartAudit	Director Técnico	santiago.ruiz@email.com	+34 555 7890

II. Control de Versiones

Versión	Fecha	Descripción	Autor	Revisado Por

2. Scope

El presente documento de auditoría está circunscrito exclusivamente al análisis y evaluación de los siguiente Smart Contract ; **iebs_Faillapop_DAO.sol**, **iebs_Faillapop_ERC20.sol**, **iebs_Faillapop_shop.sol** y **iebs_Faillapop_vault.sol** y sus interface ; **IFP_DAO.sol**, **IFP_NFT.sol**, **IFP_Shop.sol** y **IFP_Vault.sol**.

El alcance de esta auditoría se limita a las versiones de los contratos alojadas en el repositorio indicado, accedibles a través del enlace **https://github.com/jcontrerasd/Proyecto_Final_Vulnerabilidades**, y se restringe al **commit ID be079b518d1d03d739719f38e639de719a6ff1ee**.

Es importante enfatizar que no se incluirán en esta auditoría otros repositorios ni contratos adicionales que no se hayan mencionado en este documento. Cualquier extensión del alcance, que involucre la revisión de repositorios adicionales o contratos no enlistados previamente, requerirá una actualización formal del documento de auditoría, así como una aprobación explícita de todas las partes interesadas.

Los hashes proporcionados corresponden a los estados de los archivos en el momento exacto del commit especificado y sirven como referencia inmutable para verificar la integridad de los contratos en futuras consultas o auditorías.

Este enfoque garantiza la precisión, la relevancia y la integridad de la auditoría, proporcionando un marco de trabajo claro y bien definido que guía el proceso de revisión y garantiza que todos los involucrados tengan una comprensión común de los límites dentro de los cuales se realiza este ejercicio de seguridad y verificación.

Detalle

- Repositorio de código: **https://github.com/jcontrerasd/Proyecto_Final_Vulnerabilidades**
- Lista de contratos a auditar:
  - **iebs_Faillapop_DAO.sol**
  - **iebs_Faillapop_ERC20.sol**
  - **iebs_Faillapop_shop.sol**
  - **iebs_Faillapop_vault.sol**
- Interfaces a auditar :
  - **IFP_DAO.sol**
  - **IFP_NFT.sol**
  - **IFP_Shop.sol**
  - **IFP_Vault.sol**
- Commit ID que ha sido auditado: **be079b518d1d03d739719f38e639de719a6ff1ee**
- Tiempo destinado a la auditoría: 2 semanas (160 Horas) por un equipo de 2 Consultores
- Hashes de los diferentes archivos en el scope (***shasum -a 256 nombre_del_archivo.sol***):
  - **iebs_Faillapop_DAO.sol : 76aa96ddb323f615ba4136c4453ebeb40ac06459954fb3186ac0b162f5520ac3**
  - **iebs_Faillapop_ERC20.sol : 40d503fa9fdb0d7638b06da84268c28684e15b10dabd12338a47084f30a48b2f**
  - **iebs_Faillapop_shop.sol : 9db212ed0239f2a614d3291adc811c4d83a4f034846027e1b41ad42fffe5b38c**
  - **iebs_Faillapop_vault.sol : c1cc8e7b41875fd314996b08d6ef3f9fe53bd455b691daaefe616317c19c895f**

- **IFP_DAO.sol : 462f22eb6d950a1905b9f241a335dd00d48e19916dad0e6d58cb75c145a0573d**
- **IFP_NFT.sol : 9d8b00d647616bff1a9580bffff395a53d708fe0fd233f4beb4413aec282ffc6**
- **IFP_Shop.sol : e728413ae14e0b1ff3c9a91eb9a35cf4cbfd1061d54ccfbf24b45872e66714e0**
- **IFP_Vault.sol : 8af0319ea5adecbabb870b83490155d5b5e30c6862963faac6773e70df90c676**

3. Resumen Ejecutivo

En nuestra revisión de seguridad de los Smart Contract que sustentan sus operaciones, hemos identificado riesgos críticos que requieren atención inmediata. Estos riesgos, si no se abordan, podrían permitir a actores malintencionados tomar el control de los contratos, manipular fondos, y alterar el comportamiento esperado de nuestras operaciones automatizadas.

Los contratos en análisis están diseñados para resolver disputas, gestionar tokens y proporcionar funcionalidades de comercio en línea. Sin embargo, se han identificado vulnerabilidades que podrían tener graves consecuencias si no se resuelven adecuadamente.

En el contrato **FP_DAO**, una vulnerabilidad de debilidad en la generación de números pseudoaleatorios (*Weak Pseudo-Randomness*) en el sistema de lotería podría permitir manipulaciones injustas. Además, problemas de inicialización de estado (*uninitialized-state*) en las votaciones pueden afectar la integridad de las decisiones tomadas.

En el contrato **FP_Token**, la vulnerabilidad de código muerto (*Dead Code*) en la función *_beforeTokenTransfer* podría conducir a problemas de funcionalidad y seguridad. Además, se han detectado problemas con la convención de nombres y la precisión de dígitos en el contrato.

En el contrato **FP_Shop**, existen múltiples vulnerabilidades, incluyendo la posibilidad de envío arbitrario de Ether, ataques de denegación de servicio, problemas de control de acceso y fallos lógicos en diversas funciones, lo que podría llevar a pérdidas financieras y disputas no resueltas.

Por último, en el contrato **FP_Vault**, se ha identificado una vulnerabilidad de *Reentrancy benigna* en la función *distributeSlashing*, lo que podría afectar la gestión de fondos y la seguridad de la plataforma.

La importancia de abordar estas vulnerabilidades radica en la integridad, la seguridad y la funcionalidad de la plataforma. De no resolverse, estas vulnerabilidades podrían dar lugar a pérdidas financieras, disputas no resueltas y la posibilidad de manipulación injusta en el sistema de lotería. Es esencial tomar medidas inmediatas para solucionar estas vulnerabilidades y garantizar la confianza de los usuarios en la plataforma.

Es importante notar que la seguridad no es un estado, sino un proceso continuo. Por lo tanto, recomendamos una revisión y actualización regulares de sus sistemas para protegernos contra amenazas emergentes. La inversión en la seguridad de sus contratos inteligentes es esencial para mantener la confianza de sus usuarios/clientes y la integridad de la plataforma.

La acción inmediata para abordar estos problemas asegurará que la infraestructura siga siendo sólida, segura y confiable.

4. Criterios de Clasificación

La siguiente tabla proporciona una estructura para evaluar y clasificar las vulnerabilidades encontradas durante auditorías de Smart Contract y las cuales hacen parte de nuestra metodología. Estos niveles de riesgo, desde crítico hasta bajo, ayudan a determinar la prioridad y la urgencia de las respuestas de mitigación necesarias. Las descripciones resumen el impacto potencial de cada tipo de vulnerabilidad, proporcionando un marco claro para la acción correctiva y preventiva en el desarrollo y mantenimiento de contratos inteligentes seguros.

Nivel de Riesgo	Descripción
Crítico	Vulnerabilidades que permiten atacantes drenar fondos o tomar control total del contrato.
Alto	Defectos que exponen a manipulaciones de transacciones o a pérdidas financieras moderadas.
Medio	Problemas que pueden conducir a comportamientos imprevistos sin comprometer directamente los fondos.
Bajo	Inconsistencias menores que tienen un impacto limitado y no afectan la seguridad financiera.

5. Disclaimers

El cliente reconoce que, a pesar de nuestros mejores esfuerzos y experiencia en auditoría de Smart Contract, el proceso de identificación de vulnerabilidades es inherentemente complejo y no puede garantizar la detección de todas las posibles amenazas. La consultoría de auditoría no asume responsabilidad por cualquier vulnerabilidad no detectada o cualquier perjuicio que pueda derivarse de ella. Nuestro objetivo es minimizar los riesgos al máximo, pero no podemos garantizar una cobertura absoluta. Se recomienda encarecidamente al cliente adoptar medidas adicionales de seguridad y diligencia debida para proteger sus activos digitales.

Por medio de esta auditoría de seguridad y análisis, hemos identificado diversas vulnerabilidades en el sistema evaluado. Es importante destacar que las modificaciones y ajustes sugeridos para resolver estas vulnerabilidades se presentan exclusivamente como recomendaciones para el cliente. La consultoría de seguridad y auditoría proporciona orientación y asesoramiento experto en la identificación de posibles riesgos y soluciones, pero la implementación de dichas recomendaciones y mejoras debe ser responsabilidad exclusiva del cliente así como su puesta en producción. Estas sugerencias tienen como objetivo mejorar la seguridad y la integridad del sistema, y su adopción es fundamental para mantener la robustez y la confiabilidad del mismo.

6. SC0001-iebs_Faillapop_DAO.sol

Descripción General del Contrato *(/contracts/iebs_Faillapop_DAO.sol)*

El contrato FP_DAO es un sistema de votación basado en tokens ERC20 para resolver disputas en una plataforma de comercio. Los usuarios votan en disputas entre compradores y vendedores utilizando tokens FPT, y las decisiones se toman basándose en la mayoría de votos. Cada disputa tiene un identificador único y contiene argumentos de ambas partes, junto con el recuento de votos. El contrato interactúa con contratos externos ; Shop y NFT para gestionar las consecuencias de las disputas. Además, incluye un sistema de lotería que premia a los usuarios con NFTs si votan por el lado ganador. El acceso a funciones clave está protegido por una contraseña, y solo el contrato de la tienda puede iniciar o cancelar disputas.

I. Vulnerabilidades Identificadas

1. SC0001-001 WEAK PSEUDO-RANDOMNESS : LOTTERYNFT

Descripción: La función "**lotteryNFT**" intenta calcular un número aleatorio para entregar un NFT a los participantes. El algoritmo usa datos conocidos públicamente, con lo que se podrá predecir si un usuario conseguirá NFT premium en un bloque o no.

Criticidad: Alta

Impacto: El impacto principal es la previsibilidad de los resultados. Un atacante con conocimiento suficiente de los datos del blockchain podría anticipar o influir en los resultados de la lotería para obtener NFT premium de manera desproporcionada, comprometiendo la imparcialidad del juego y posiblemente llevando a pérdidas financieras para otros participantes.

Código Original: *(/contracts/iebs_Faillapop_DAO.sol :251-267)*

```
247     /**
248         @notice Run a PRNG to award NFT to a user
249         @param user The address of the elegible user
250      */
251     function lotteryNFT(address user) internal {
252         uint randomNumber = uint8(
253             uint256(
254                 keccak256(
255                     abi.encodePacked(
256                         blockhash(block.number - 1),
257                         block.timestamp,
258                         user
259     ))));
260
261         if (randomNumber < THRESHOLD) {
262             nftContract.mintCoolNFT(user);
263
264         }
265
266         emit AwardNFT(user);
267     }
```

Cómo Explotarlo : Un atacante podría crear un smart contract malicioso y esperar hasta que llegue un bloque en el que pueda recibir NFT premium para llamar a "checkLottery".

Paso 1: Monitoreo de Bloques
El atacante monitorea los bloques y calcula el resultado del número pseudoaleatorio basado en el blockhash del bloque anterior, el timestamp del bloque actual, y su propia dirección.

Paso 2: Despliega un contrato inteligente malicioso
Está programado para llamar a la función checkLottery en el momento preciso en que las condiciones del bloque coinciden con un resultado favorable.

Paso 3: Ejecución en el Momento Óptimo
El contrato malicioso ejecuta la llamada a checkLottery en el bloque específico donde la predicción del número pseudoaleatorio indica una alta probabilidad de ganar un NFT premium.

Explicación Mitigación: Para mitigar esta vulnerabilidad, se recomienda utilizar un oráculo para generar números aleatorios o implementar un mecanismo de compromiso-revelación. Evitar el uso de variables predecibles como block.timestamp y blockhash para la generación de números aleatorios, ya que estos pueden ser manipulados o anticipados por los mineros o los atacantes.

Código Corregido: (https://docs.chain.link/vrf/v1/best-practices)

```
import "@chainlink/contracts/src/v0.8/VRFConsumerBase.sol";

contract VulnerableDAO is VRFConsumerBase {

    bytes32 internal keyHash; // Identificador de la clave de Chainlink VRF
    uint256 internal fee; // Costo en LINK para solicitar un número aleatorio
    mapping(bytes32 => address) private requestToUser; // Mapeo de solicitud de número aleatorio a la dirección del usuario
    // Eventos para registrar solicitudes y entrega de NFT
    event RandomNumberRequested(bytes32 indexed requestId, address indexed user);
    event AwardNFT(address indexed user, uint256 randomNumber);

    // Constructor para inicializar el contrato con la configuración de Chainlink VRF
    constructor(address vrfCoordinator, address linkToken, bytes32 _keyHash, uint256 _fee)
    VRFConsumerBase(vrfCoordinator, linkToken) {
        keyHash = _keyHash;
        fee = _fee;
    }

    // Función para verificar la posibilidad de ganar un NFT en una lotería
    function checkLottery(uint disputeID) external {
        // Solicitar un número aleatorio a Chainlink VRF
        bytes32 requestId = requestRandomness(keyHash, fee);
        requestToUser[requestId] = msg.sender; // Asociar la solicitud con el usuario
        emit RandomNumberRequested(requestId, msg.sender); // Emitir evento de solicitud
    }
    // Sobrescritura de fulfillRandomness para manejar el número aleatorio recibido de Chainlink VRF
    function fulfillRandomness(bytes32 requestId, uint256 randomness) internal override {
        address user = requestToUser[requestId]; // Obtener el usuario asociado con la solicitud
        lotteryNFT(user, randomness); // Llamar a la función de lotería con el número aleatorio
        delete requestToUser[requestId]; // Limpiar el mapeo después de procesar la solicitud
    }
// Función para determinar si se otorga un NFT basado en el número aleatorio

function lotteryNFT(address user, uint256 randomness) internal {
    if (randomness < THRESHOLD) {
        // Actualizar el estado aquí si es necesario, antes de la llamada externa
        // Por ejemplo, marcar que el NFT ha sido otorgado
        nftAwarded[user] = true;
       // Emitir el evento antes de la llamada externa
```

```
        emit AwardNFT(user, randomness);
        // Luego, realizar la llamada externa
        // La llamada externa se hace después de actualizar el estado y emitir el evento
        nftContract.mintCoolNFT(user);
    }
}
```

IMPORTANTE : La variable nftAwarded debería ser una variable de estado mapeada en el contrato para rastrear si se ha otorgado un NFT al usuario. Esto no solo ayuda a prevenir la Reentrancy, sino que también evita que un usuario reciba más de un NFT debido a múltiples llamadas a la función lotteryNFT.

2. SC0001-002 UNINITIALIZED-STATE : QUORUM

Descripción: La variable **quorum** del contrato **FP_DAO**, que establece el mínimo de votantes para resolver disputas, no está inicializada en el constructor ni en cualquier otro lugar. Esto podría ocasionar que la función **endDispute**(uint256) opere con un valor de **quorum** indeterminado, afectando la lógica de cierre de disputas.

Criticidad: Alta

Impacto: La falta de inicialización del **quorum** permite resoluciones de disputas sin el consenso adecuado, poniendo en riesgo la imparcialidad y la seguridad del mecanismo de votación, lo que podría resultar en decisiones arbitrarias y pérdida de confianza en el sistema.

Código Original: *(/contracts/iebs_Faillapop_DAO.sol:63)*

```
62      ///@notice Min number of people to pass a vote
63      uint256 quorum;
```

Cómo Explotarlo : La explotación de una variable no inicializada como lo es en este caso **quorum** podría realizarse si un actor malintencionado encuentra que este valor no está establecido y, por lo tanto, el contrato no verifica adecuadamente las condiciones antes de ejecutar acciones críticas. Un atacante podría influir en la resolución de disputas con un número insuficiente de votos, lo que podría alterar los resultados de la votación a su favor con pocos o ningún voto legítimo.

Paso 1: Análisis del Contrato
El atacante revisaría el código del contrato inteligente para entender su lógica y determinar que la variable quorum no está inicializada.

Paso 2: Monitoreo de Actividad
Monitorearían las disputas abiertas y las votaciones en curso para identificar oportunidades donde el número de votos emitidos es bajo o cuando es más probable que la función endDispute sea llamada.

Paso 3: Creación de una Disputa
El atacante podría abrir una disputa o esperar a que una esté próxima a cerrarse. Si quorum es cero o indefinido, cualquier número de votos podría ser suficiente para decidir el resultado.

Paso 4: Votación Estratégica
El atacante emitiría votos para la disputa. Si el contrato no valida adecuadamente el quorum, incluso un solo voto podría ser suficiente para influir en el resultado.

Paso 5: Explotación del Resultado
Una vez que la votación es influida, el atacante podría beneficiarse del resultado. Por ejemplo, si la disputa involucra la transferencia de fondos o activos, podrían direccionarlos de manera inapropiada.

Explicación Mitigación: Para mitigar la vulnerabilidad, se debe inicializar la variable quorum dentro del constructor del contrato con un valor predefinido, asegurando así que exista un umbral de votantes para validar las resoluciones de disputas. Esto garantiza la consistencia y la integridad del proceso de votación.

Código Corregido:

```
   //  DEFAULT_QUORUM es una constante que se debe definir en el contrato para representar el quórum predeterminado.

   quorum = DEFAULT_QUORUM;

 constructor(string memory magicWord, address shop, address nft_addr, address fpt_addr) {
       password = magicWord;
       shop_addr = shop;
       shopContract = IFP_Shop(shop_addr);
       nftContract = IFP_NFT(nft_addr);
       fptContract = IERC20(fpt_addr);
// Inicialización de la variable quorum.
// initialQuorum sería el valor que se pasa al contrato cuando se despliega, permitiendo a quien lo despliegue establecer el quórum inicial.
       quorum = initialQuorum;
    }
```

7. SC0002- iebs_Faillapop_ERC20.sol

Descripción General del Contrato *(/contracts/iebs_Faillapop_ERC20.sol)*

El contrato **FP_Token** es un token **ERC20** personalizado con funcionalidades adicionales para la gobernanza. Incorpora características de quemado de tokens (ERC20Burnable), pausabilidad (Pausable) y control de acceso (AccessControl). El contrato permite que el poseedor del rol PAUSER_ROLE pueda pausar y reanudar las transferencias de tokens, y el poseedor del rol MINTER_ROLE tiene la capacidad de acuñar (mint) nuevos tokens. Al desplegarse, el contrato asigna roles de administrador, pausador y acuñador al creador del contrato y emite una cantidad inicial de tokens. La función *_beforeTokenTransfer* asegura que las transferencias de tokens solo ocurran cuando el contrato no esté pausado, pero parece contener un error en su implementación actual.

I. Vulnerabilidades Detectadas

1. SC0002-001 DEAD CODE : _BEFORETOKENTRANSFER

Descripción: La La función *_beforeTokenTransfer* en el contrato **FP_Token** está definida pero nunca se utiliza en el flujo del contrato. Esto indica una funcionalidad no implementada o residual.

Criticidad: Bajo

Impacto: Código muerto como _beforeTokenTransfer puede llevar a confusión y mantenimiento ineficiente del código. Su eliminación aclararía la funcionalidad del contrato.

Código Original: *(/contracts/iebs_Faillapop_ERC20:36-43)*

```
36      function _beforeTokenTransfer(address from, address to, uint256 amount)
37          internal
38          whenNotPaused
39          override
40      {
41          super._beforeTokenTransfer(from, to, amount);
42
43      }
```

Cómo Explotarlo : Esta vulnerabilidad no es directamente explotable ya que no presenta un riesgo de seguridad inmediato. En lugar de explotarla, un atacante podría buscar códigos muertos para comprender la lógica interna y buscar errores relacionados.

Explicación Mitigación: Eliminar el código muerto y revisar el contrato para otras funciones no utilizadas mejora la claridad y reduce la superficie de ataque.

2. SC0002-002 NAMING CONVENTION : FP_TOKEN

Descripción: El contrato **FP_Token** no sigue la convención de nomenclatura estándar de Solidity CapWords, afectando la consistencia y legibilidad del código.

Criticidad: Bajo

Impacto: Ignorar las convenciones puede no afectar la funcionalidad, pero afecta la legibilidad y puede causar confusión o errores en el desarrollo futuro.

Código Original: ***(/contracts/ iebs_Faillapop_ERC20:13)***

```
13  contract FP_Token is ERC20, ERC20Burnable, Pausable, AccessControl {
14      bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
15      bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
16
17      constructor() ERC20("FaillaPop Token", "FPT") {
18          _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
19          _grantRole(PAUSER_ROLE, msg.sender);
20          _mint(msg.sender, 1000000 * 10 ** decimals());
21          _grantRole(MINTER_ROLE, msg.sender);
22      }
23
```

Cómo Explotarlo : La explotación de una convención de nomenclatura no estándar no es aplicable, ya que es una cuestión de estilo y no una vulnerabilidad de seguridad. Sin embargo, podría indicar falta de atención a detalles que podría explorarse para buscar deficiencias más críticas.

Explicación Mitigación: Aplicar la convención de nomenclatura CapWords a todos los contratos y sus elementos para mantener el estándar y facilitar la comprensión.

3. SC0002-003 TOO MANY DIGITS : FP_TOKEN

Descripción: En el constructor de **FP_Token**, se usan literales numéricos con muchos dígitos sin separadores, lo que puede llevar a errores de interpretación.

Criticidad: Bajo

Impacto: El uso de literales con muchos dígitos podría causar confusión y errores en la gestión del contrato. Simplificar estos números mejoraría la claridad.

Código Original: ***(/contracts/ iebs_Faillapop_ERC20:17-22)***

```
13  contract FP_Token is ERC20, ERC20Burnable, Pausable, AccessControl {
14      bytes32 public constant PAUSER_ROLE = keccak256("PAUSER_ROLE");
15      bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
16
17      constructor() ERC20("FaillaPop Token", "FPT") {
18          _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
19          _grantRole(PAUSER_ROLE, msg.sender);
20          _mint(msg.sender, 1000000 * 10 ** decimals());
21          _grantRole(MINTER_ROLE, msg.sender);
22      }
23
```

Cómo Explotarlo : Aunque no es una vulnerabilidad que se pueda "explotar" en el sentido convencional, el entendimiento incorrecto de literales numéricos podría ser aprovechado para inducir a error en el mantenimiento o en procesos de auditoría.

Explicación Mitigación: Utilizar separadores numéricos o expresiones más claras para grandes literales mejora la legibilidad y reduce los errores humanos.

8. SC0003- iebs_Faillapop_shop.sol

Descripción General del Contrato *(/contracts/iebs_Faillapop_shop.sol)*

El contrato **FP_Shop** es una plataforma de marketplace en Blockchain que permite a los usuarios vender y comprar bienes utilizando Ether. Los vendedores deben bloquear fondos para evitar comportamientos maliciosos y los compradores pueden abrir disputas si están insatisfechos, las cuales son resueltas por un DAO. Incluye funciones para crear, modificar, comprar y cancelar ventas, así como para gestionar disputas y devoluciones. Los vendedores pueden ser sancionados o incluidos en una lista negra si se comportan de manera maliciosa. El contrato interactúa con contratos de DAO y bóveda (Vault) para la gobernanza y la seguridad de los fondos, respectivamente. Además, implementa roles de control de acceso para administrar diferentes privilegios dentro del contrato.

I. Vulnerabilidades Detectadas

1. SC0003-001 ARBITRARY-SEND-ETH : REIMBURSE

Descripción: La función **reimburse** permite a cualquier usuario solicitar un reembolso sobre cualquier venta activa en el marketplace sin realizar ninguna validación sobre la identidad del caller o el estado legítimo de reembolso de la venta. Esto permite que un atacante robe fondos arbitrarios del marketplace llamando a reimburse desde su propia dirección.

Criticidad: Crítico

Impacto: Esta Esta vulnerabilidad permite el robo de fondos del contrato por parte de usuarios no autorizados, pudiendo vaciar por completo los fondos del marketplace y de los vendedores, resultando en pérdidas financieras y desconfianza en la plataforma.legítimas.

Código Original: *(/contracts/iebs_Faillapop_shop.sol:377-386)*

```
373     /**
374         @notice Reimburst a buyer.
375         @param itemId The ID of the item being reimbursed
376     */
377     function reimburse(uint itemId) internal {
378         uint price = offered_items[itemId].price;
379         address buyer = offered_items[itemId].buyer;
380 
381         // Pay the buyer back
382         (bool success, ) = payable(buyer).call{value: price}("");
383         require(success, "Sale payment failed");
384 
385         emit Reimburse(buyer, price);
386     }
```

Cómo Explotarlo : Un atacante podría explotar un contrato inteligente vulnerable seleccionando un itemId de una venta activa y llamando a la función reimburse sin validaciones de seguridad. Esto le permitiría recibir fondos indebidamente, resultando en un robo si la función no verifica la legitimidad de la solicitud de reembolso.

Paso 1: Seleccionar itemId
Un atacante buscaría un itemId de una venta activa en el marketplace. Esto podría hacerse observando las transacciones en la cadena de bloques que interactúan con el contrato en cuestión o accediendo al estado del contrato a través de un nodo de Ethereum**.**

Paso 2: Llamar a reimburse
El atacante, a continuación, invocaría la función reimburse del contrato inteligente, pasando el itemId obtenido. Si no hay validaciones adecuadas en la función, esta invocación podría realizarse desde cualquier cuenta, no solo desde la del comprador o un administrador autorizado.

Paso 3: Recibir fondos
Dado que no hay validaciones, la función reimburse ejecutaría el reembolso y los fondos asociados con la venta serían transferidos a la dirección del llamador (caller). En un contrato vulnerable, esto sucedería sin verificar si el llamador tiene derecho a los fondos.

Paso 4: Robar fondos
Los fondos serían transferidos exitosamente a la cuenta del atacante. En un entorno real, esto constituiría un robo de fondos.

Explicación Mitigación: Se debe validar que sólo el comprador o el admin puedan llamar a reimburse y que el itemId exista y esté en disputa o pendiente para prevenir robo de fondos, por lo que es esencial implementar controles que verifiquen la validez y el estado de los itemIds en ventas activas, confirmen los permisos del solicitante y monitoreen las transacciones para detectar anomalías, asegurando así la legitimidad y seguridad en las operaciones de reembolso.

Código Corregido:

```
    @notice Reembolsar a un comprador.
    @param itemId El ID del artículo que se reembolsa.
 */
function reimburse(uint itemId) public {
    // Validación de existencia del item y que esté en estado de disputa o pendiente
    require(offered_items[itemId].state == State.Disputa || offered_items[itemId].state == State.Pendiente, "El item no esta en disputa o pendiente");

    uint price = offered_items[itemId].price;
    address buyer = offered_items[itemId].buyer;
    // Verificar que el caller sea el comprador o un administrador
    require(msg.sender == buyer || msg.sender == admin, "Caller no autorizado");
    // Proceder con el reembolso
    (bool success, ) = payable(buyer).call{value: price}("");
    require(success, "Sale payment failed");
    // Actualizar el estado del item
    offered_items[itemId].state = State.Reembolsado;
    emit Reimburse(buyer, price);
}
```

2. SC0003-002 DENIAL-OF-SERVICE : SETVACATIONMODE

Descripción: La función **setVacationMode** permite a un atacante cambiar el estado de cualquier venta activa a "Vacation" sin realizar ninguna validación sobre si el caller es realmente el vendedor. Esto permite realizar ataques de denegación de servicio.

Criticidad: Alto

Impacto: Un atacante podría deshabilitar ventas arbitrarias dentro del marketplace mediante DoS, resultando en pérdida de ingresos para vendedores legítimos y una mala reputación para la plataforma.

Código Original: *(/contracts/iebs_Faillapop_shop.sol:272-285)*

```
268     /**
269         @notice Endpoint to set the vacation mode of a seller. If the seller is in vacation mode nobody can buy his goods
270         @param _vacationMode The new vacation mode of the seller
271      */
272     function setVacationMode(bool _vacationMode) external {
273         for (uint i = 0; i < offerIndex; i++) {
274             if (offered_items[i].seller == msg.sender) {
275 
276                 if (_vacationMode && offered_items[i].state == State.Selling) {
277                     offered_items[i].state = State.Vacation;
278 
279                 } else if (!_vacationMode && offered_items[i].state == State.Vacation) {
280                     offered_items[i].state = State.Selling;
281 
282                 }
283             }
284         }
285     }
```

Cómo Explotarlo : La función setVacationMode en su estado actual no restringe el cambio de estado a "Vacation" solo al vendedor del artículo, permitiendo que cualquier usuario invoque la función y cambie el estado de todas las ventas activas del vendedor que hace la llamada. Esto puede ser explotado por un atacante para bloquear todas las ventas de un vendedor, interrumpiendo las operaciones normales del marketplace y causando pérdidas financieras y daño a la reputación de la plataforma..

Paso 1: Selección de un itemId Válido
El atacante identifica un itemId correspondiente a una venta activa en el estado "Selling" mediante el análisis de las transacciones en la cadena de bloques o el estado del contrato.

Paso 2: Invocación de setVacationMode
Utilizando el itemId seleccionado, el atacante llama a la función setVacationMode pasando true como argumento para activar el modo de vacaciones.

Paso 3: Bloqueo de Ventas
Sin una validación adecuada, la función aplica el modo de vacaciones a todas las ventas del vendedor que hace la llamada, sin importar si quien llama es el verdadero vendedor del itemId.

Paso 4: Realización de DoS
Como resultado, las ventas quedan bloqueadas en modo de vacaciones, efectivamente realizando un ataque de DoS al vendedor legítimo.

Explicación Mitigación: Para fortalecer la seguridad y evitar ataques de denegación de servicio (DoS), se deben implementar controles de acceso estrictos en la función setVacationMode. Se debería introducir una validación que confirme que solo el vendedor registrado de un artículo pueda cambiar su estado a modo de vacaciones.

Código Corregido:

```
// Debe existir un mapeo para rastrear si una dirección es un vendedor
mapping(address => bool) public sellers;

/**
 * @notice Verifica si la dirección proporcionada es un vendedor.
 * @param _address La dirección a verificar.
 * @return bool Verdadero si la dirección es un vendedor, falso en caso contrario.
 */
function isSeller(address _address) public view returns (bool) {
    return sellers[_address];
}

/**
 * @notice Endpoint para establecer el modo de vacaciones de un vendedor. Si el vendedor está en modo de
vacaciones, nadie puede comprar sus productos.
 * @param _vacationMode El nuevo modo de vacaciones del vendedor.
 */
function setVacationMode(bool _vacationMode) external {
    require(isSeller(msg.sender), " Sólo el vendedor puede configurar el modo vacaciones ");

    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;
            }
        }
    }
}
```

3. SC0003-003 ACCESS-CONTROL : REMOVEMALICIOUSSALE

Descripción: La función **removeMaliciousSale** presenta una vulnerabilidad crítica de seguridad al otorgar a los administradores el poder de cancelar ventas pendientes y apoderarse de los fondos. Sin las salvaguardas adecuadas, esta capacidad puede ser explotada para desviar fondos que deberían ser liberados a los vendedores legítimos tras la conclusión de una venta.

Criticidad: Crítico

Impacto: El impacto indica que si un administrador se viese comprometido, podría abusar de la función para desviar fondos de ventas legítimas, lo que conllevaría a pérdidas económicas directas

para los vendedores. Además, tal incidente expondría la confianza en la plataforma, pudiendo dañar

```
317     /**
318         @notice Endpoint to remove a malicious sale and slash the stake. The owner of the contract can remove a malicious sale and blacklist the seller
319         @param itemId The ID of the item which sale is considered malicious
320      */
321     function removeMaliciousSale(uint itemId) external onlyRole(ADMIN_ROLE) {
322         require(offered_items[itemId].seller != address(0), "itemId does not exist");
323 
324         if (offered_items[itemId].state == State.Pending) {
325             reimburse(itemId);
326         } else if (offered_items[itemId].state == State.Disputed) {
327             reimburse(itemId);
328             closeDispute(itemId);
329         }
330 
331         // Seller should NOT be paid
332         closeSale(itemId, false);
333         blacklist(offered_items[itemId].seller);
334     }
```

su reputación y viabilidad a largo plazo.

Código Original: *(/contracts/iebs_Faillapop_shop.sol:321-334)*

Cómo Explotarlo : Un administrador malintencionado podría explotar esta función identificando una venta en estado pendiente y ejecutando removeMaliciousSale para cerrar la venta. Esto cancelaría la transacción, liberaría los fondos y permitiría al administrador desviar indebidamente el dinero destinado al vendedor.

Paso 1: Identificar una Venta en Estado Pendiente
Un administrador malicioso rastrea las ventas hasta encontrar una con el estado Pending, probablemente monitoreando las transacciones o consultando el estado del contrato.

Paso 2: Ejecución de removeMaliciousSale como Administrador
El administrador usa sus privilegios para llamar a removeMaliciousSale, pasando el itemId de la venta identificada.

Paso 3: Cierre Forzado de la Venta
La función termina la venta sin ejecutar el pago al vendedor legítimo. En lugar de proteger contra el comportamiento malicioso, se abusa de ella para controlar los fondos.

Paso 4: Desvío de Fondos
Los fondos que estaban destinados al vendedor se desbloquean del contrato de Vault y, en el proceso, se desvían de manera indebida, permitiendo que el administrador malintencionado se apropie de ellos.

Explicación Mitigación: Para mitigar la vulnerabilidad y prevenir el robo de fondos, la función removeMaliciousSale debe restringirse para que solo se ejecute bajo circunstancias específicas. Primero, debe verificarse que la venta esté actualmente en disputa o ya haya concluido, lo que implicaría la adición de controles que aseguren que la venta no esté en un estado activo o pendiente. Así, se evitaría que administradores malintencionados o comprometidos cancelen arbitrariamente transacciones en curso y desvíen fondos. Estas restricciones protegerían los intereses de los vendedores honestos y mantendrían la integridad financiera y la confianza en la plataforma.

Código Corregido:

```
} function removeMaliciousSale(uint itemId) external onlyRole(ADMIN_ROLE) {
    require(
        offered_items[itemId].state == State.Disputed ||
        offered_items[itemId].state == State.Sold,
        " No se puede eliminar: la venta no está en estado extraíble."
```

```
    );

    // En caso de que la venta esté en pendiente, se devuelve el dinero al comprador
    if (offered_items[itemId].state == State.Pending) {
        reimburse(itemId);
    }

    // Eliminar la venta y liberar los fondos bloqueados del vendedor si es apropiado
    closeSale(itemId, false);

    // Adicionalmente, si el vendedor es malicioso, puede ser añadido a la lista negra
    blacklist(offered_items[itemId].seller);
}
```

4. SC0003-004 LOGIC FAILURE : DOBUY

Descripción: La función doBuy() permite a un usuario comprar un item enviando más ether del precio establecido, sin embargo, el contrato no cuenta con una funcionabilidad para que el usuario recupere posteriormente el ether adicional enviado por error..

Criticidad: Bajo

Impacto: Esta vulnerabilidad resultaría en la pérdida definitiva de fondos por parte de usuarios que por error envíen más ether del necesario para realizar una compra. Esto generaría insatisfacción en los clientes y una mala reputación para el marketplace.

Código Original:

```
133     /**
134         @notice Endpoint to buy an item
135         @param itemId The ID of the item being bought
136         @dev The user must send the exact amount of Ether to buy the item
137      */
138     function doBuy(uint itemId) external payable {
139         require(offered_items[itemId].seller != address(0), "itemId does not exist");
140         require(offered_items[itemId].state == State.Selling, "Item cannot be bought");
141         require(msg.value >= offered_items[itemId].price, "Incorrect amount of Ether sent");
142
143         offered_items[itemId].buyer = msg.sender;
144         offered_items[itemId].state = State.Pending;
145
146         emit Buy(msg.sender, itemId);
147     }
```

(/contracts/iebs_Faillapop_shop.sol:138-147)

Cómo Explotarlo : La función doBuy permite a un usuario comprar un item enviando más ether del precio establecido, pero al no existir una funcionalidad de reembolso, no hay forma de recuperar el dinero adicional enviado por error.

Paso 1: Identificar item de alto valor
Obtener el ID de un item con un precio cualquiera, por ejemplo de 2 eth

Paso 2: Ejecutar compra con monto erroneo
Llamar a la función doBuy del contrato enviando el ID del item, y el monto de pago superior al precio base, 2.5 eth

Paso 3: Fondos perdidos permanentemente
Validar que la compra se haya realizado correctamente. Los 0.5 eth adicionales quedan atrapados en el contrato de forma permanente sin ninguna funcionabilidad de recuperación.

Explicación Mitigación: Para mitigar la vulnerabilidad, es clave implementar una comprobación en la función doBuy que verifique que el ether enviado es igual al precio del artículo listado. Esta validación debe ocurrir antes de que la transacción se considere exitosa, asegurando así que el vendedor reciba el pago completo y que el comprador pago lo justo.

```
/**
     @notice Endpoint to buy an item
     @param itemId The ID of the item being bought
     @dev The user must send the exact amount of Ether to buy the item
   */
  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);
  }
```

5. SC0003-005 ARBITRARY-SEND-ETH : CLOSESALE

Descripción: La función **closeSale** puede ser llamada por cualquier usuario que pueda manipular el estado de toBePaid para forzar un pago al vendedor sin haberse completado la venta, debido a la falta de verificaciones adecuadas de quién puede ejecutar la función y en qué condiciones.

Criticidad: Medio

Impacto: Al explotar esta vulnerabilidad, los vendedores pueden cobrar indebidamente por artículos no entregados. Esto resulta en que los compradores pierdan Ether sin recibir nada a cambio, socavando la confianza y la integridad de la plataforma.

Código Original: ***(/contracts/iebs_Faillapop_shop.sol:344-357)***

Cómo Explotarlo : Para explotar la vulnerabilidad en closeSale, un atacante podría identificar una venta en estado manipulable (como Pending o Disputed), y luego invocar indebidamente closeSale, forzando el cierre y el pago al vendedor sin que la transacción se haya completado legítimamente. Este abuso de la función resulta en transferencias de fondos no autorizadas, perjudicando a los compradores y comprometiendo la integridad de la plataforma.

Paso 1: Identificación de la Venta
El atacante busca en el contrato un itemId que esté en un estado manipulable como Pending o Disputed. Este paso es crucial para encontrar una venta que pueda ser explotada sin haberse completado legítimamente.

Paso 2: Forzar el Cierre de la Venta
Invocar closeSale indebidamente a través de un acceso permitido (o no restringido) a la función closeSale, el atacante pasa el itemId identificado y establece toBePaid como true, intentando forzar el cierre de la venta y el desembolso de fondos al vendedor.

Paso 3: Recepción de Fondos por el Vendedor
El contrato procesa la solicitud y realiza un pago al vendedor, a pesar de que la venta no ha cumplido con los requisitos legítimos de conclusión. Esto resulta en un desembolso injustificado, aprovechando la falta de validaciones en el contrato.

Explicación Mitigación: Para mitigar esta vulnerabilidad, se debe restringir la función closeSale para que solo pueda ser llamada por funciones internas después de las debidas validaciones, o implementar modificadores que verifiquen que el estado de la venta justifica el cierre y el pago. Además, se debe verificar que la persona que llama tenga la autoridad para cerrar la venta, como el comprador o el sistema de DAO al resolver una disputa..

```
339     /**
340         @notice Remove a sale from the list
341         @param itemId The ID of the item which sale is being removed
342         @param toBePaid If the seller should be paid or not
343      */
344     function closeSale(uint itemId, bool toBePaid) public {
345         address seller = offered_items[itemId].seller;
346         uint256 price = offered_items[itemId].price;
347 
348         // Seller payment
349         if (toBePaid) {
350             (bool success, ) = payable(seller).call{value: price}("");
351             require(success, "Sale payment failed");
352         }
353         // Seller stake release
354         vaultContract.doUnlock(seller, price);
355 
356         delete offered_items[itemId];
357     }
```

```
/**
    @notice Remove a sale from the list
    @param itemId The ID of the item which sale is being removed
```

```
    @param toBePaid If the seller should be paid or not
 */
function closeSale(uint itemId, bool toBePaid) internal {
    address seller = offered_items[itemId].seller;
    uint256 price = offered_items[itemId].price;
    // Verificar que la venta esté en un estado que permita su cierre
    require(
        offered_items[itemId].state == State.Sold ||
        offered_items[itemId].state == State.Disputed,
        "Venta no está en condiciones de cerrarse "
    );
    // Proceder con el pago al vendedor si es apropiado
    if (toBePaid) {
        (bool success, ) = payable(seller).call{value: price}("");
        require(success, "Sale payment failed");
    }
    // Liberar los fondos del vendedor
    vaultContract.doUnlock(seller, price);
        // Eliminar la venta del registro
    delete offered_items[itemId];
}
```

9. SC0004- iebs_Faillapop_vault.sol

Descripción General del Contrato *(/contracts/iebs_Faillapop_vault.sol)*

El contrato **FP_Vault** es un sistema de gestión de fondos. Permite a los usuarios depositar (stake) y retirar (unstake) Ether para ser utilizado en transacciones de venta. Durante una venta, los fondos del vendedor se bloquean para asegurar la transacción y prevenir comportamientos maliciosos. En caso de que un usuario sea considerado malicioso por el DAO, sus fondos pueden ser confiscados (slashed). Además, el contrato interactúa con un los contratos DAO y Shop para la gobernanza y la operación de las ventas, respectivamente. También gestiona un sistema de recompensas para usuarios privilegiados, distribuyendo Ether obtenido de los fondos confiscados. El contrato implementa roles de control de acceso para administrar diversas funcionalidades y asegura la seguridad y la buena conducta en las transacciones de la plataforma.

I. Vulnerabilidades Detectadas

1. SC0004-001 REENTRANCY-BENIGN : DISTRIBUTESLASHINGDE

Descripción: La función **distributeSlashing** está diseñada para actualizar el monto máximo que los usuarios con privilegios pueden reclamar, calculado a partir de los fondos totales confiscados. A pesar de ser una función interna, su ejecución podría ser inadvertidamente reactivada dentro de una misma transacción si otras funciones interactúan con contratos externos. Dicha Reentrancy no anticipada puede llevar a recalculos concurrentes de max_claimable_amount, generando inconsistencias en los valores destinados a los usuarios elegibles. Este escenario es particularmente crítico al revisar la integridad de las operaciones de contrato inteligente y los efectos colaterales de llamadas entre funciones.

Criticidad: Bajo

Impacto: La Reentrancy en esta función podría provocar cálculos incorrectos de recompensas, pero su impacto es limitado ya que no implica transferencia de fondos. Únicamente afecta la precisión de las recompensas, sin amenazar directamente los activos del sistema. A pesar de esto, es crucial abordar y solucionar la vulnerabilidad para mantener la integridad del sistema.

Código Original: *(/contracts/iebs_Faillapop_vault.sol:210-218)*

Cómo Explotarlo : A pesar de que la función está marcada como interna y no es accesible directamente, un llamado Reentrancy desde otra función del contrato que interactúe con contratos externos podría desencadenarla. Esto podría resultar en un recálculo no deseado del monto máximo reclamable, lo que permitiría a un atacante manipular las recompensas potenciales del sistema de manera inapropiada.

Paso 1: Activación de Reentrancy
Un atacante aprovecha una función externa del contrato que llama a distributeSlashing. Esta función es manipulada para permitir la posibilidad de una Reentrancy, lo que significa que el flujo de ejecución puede ser interrumpido y retomado en distributeSlashing.

Paso 2: Recálculo Inducido
Durante la Reentrancy, el atacante invoca nuevamente distributeSlashing. Esto desencadena una nueva ejecución de la función distributeSlashing mientras la ejecución original aún no ha terminado, lo que altera el estado del contrato de maneras no previstas originalmente.

Paso 3: Desajuste de Recompensas
La ejecución múltiple e inesperada de distributeSlashing durante la Reentrancy provoca un desajuste en los montos reclamables. Los cálculos internos de recompensas pueden verse afectados de manera incorrecta debido a la Reentrancy, lo que podría permitir al atacante manipular las recompensas potenciales del sistema.

Paso 4: Consecuencias Limitadas
Aunque la Reentrancy causa un desajuste en los cálculos de recompensas, es importante destacar que el impacto de esta explotación se mantiene en la lógica interna de cálculo de recompensas. No hay transferencia de fondos involucrada ni se comprometen directamente los activos del contrato, lo que limita el alcance de las consecuencias a la precisión de las recompensas en lugar de poner en peligro la seguridad financiera del sistema.

```
207     /************************************ Internal  ****************************************************/
208
209     ///@notice Sets a new maximum claimable amount per user based on the total slashed amount
210     function distributeSlashing(uint amount) internal {
211         totalSlashed += amount;
212
213         (, bytes memory data) = nftContract.call(abi.encodeWithSignature("totalPowersellers()"));
214         uint totalPowersellers = abi.decode(data, (uint256));
215
216         uint newMax = totalSlashed / totalPowersellers;
217         max_claimable_amount = newMax;
218     }
```

Explicación Mitigación: Para abordar esta vulnerabilidad, es esencial garantizar que la función distributeSlashing no pueda ser invocada de manera reentrante. Esto se logra mediante la aplicación de un modificador de "no Reentrancy" que evita la ejecución concurrente. Además, se debe llevar a cabo una revisión exhaustiva de todas las funciones que llaman a distributeSlashing para prevenir efectos no deseados y garantizar la seguridad en la ejecución de estas funciones.

Código Corregido:

```
// Declaración del modificador "noReentrancy"
bool private locked;

modifier noReentrancy() {
   require(!locked, "ReentrancyGuard: reentrant call");
   locked = true;
   _;
   locked = false;
}

   /********************************** Internal ***************************************/
   ///@notice Sets a new maximum claimable amount per user based on the total slashed amount
   function distributeSlashing(uint amount) public noReentrancy {
      totalSlashed += amount;
```

```
165     /**
166         @notice Modify configuration parameters, only the owner can do it
167         @param newDao The address of the new DAO contract
168         @param newShop The address of the new Shop contract
169         @param newNft The address of the new NFT contract
170      */
171     function updateConfig(address newDao, address newShop, address newNft) external onlyRole(ADMIN_ROLE) {
172         daoContract = IFP_DAO(newDao);
173         shopContract = IFP_Shop(newShop);
174         nftContract = newNft;
175
176         emit NewConfig(newDao, newShop, newNft);
177     }
```

```
      (, bytes memory data) = nftContract.call(abi.encodeWithSignature("totalPowersellers()"));
      uint totalPowersellers = abi.decode(data, (uint256));

      uint newMax = totalSlashed / totalPowersellers;
      max_claimable_amount = newMax;
   }
```

2. SC0004-002 MISSING-ZERO-CHECK : UPDATECONFIG

Descripción: La función **updateConfig** no verifica si las nuevas direcciones proporcionadas para los contratos DAO, Shop y NFT son direcciones válidas y no cero. Esto podría ser explotado para configurar direcciones nulas, deshabilitando las interacciones futuras y afectando las funcionalidades principales del contrato.

Criticidad: Alto

Impacto: Un atacante con el rol de administrador podría cambiar las direcciones a cero, interrumpiendo las operaciones esenciales del contrato, como el bloqueo y desbloqueo de fondos, lo que podría llevar a la congelación de activos y pérdida de funcionalidad.

Código Original: *(/contracts/iebs_Faillapop_vault.sol:171-177)*

Cómo Explotarlo : Un administrador con acceso privilegiado, intencionadamente o por error, podría ejecutar la función updateConfig con parámetros de direcciones cero. Esto dejaría al contrato sin referencias válidas para interacciones críticas, bloqueando efectivamente las funcionalidades de stake, reclamo de recompensas y gobernanza, resultando en un contrato no operativo y posiblemente en la pérdida de fondos.

Paso 1: Acceso Administrativo
Un atacante con privilegios de administrador, ya sea por robo de credenciales o por un rol asignado incorrectamente, posee la capacidad de modificar la configuración del contrato. Este nivel de acceso le permite cambiar parámetros clave del contrato.

Paso 2: Establecimiento de Direcciones Nulas
Utilizando su acceso administrativo, el atacante puede ejecutar la función updateConfig, pasando direcciones de contratos que son direcciones nulas (0x0). Esto puede ser un acto deliberado para desactivar las operaciones del contrato o un error debido a la falta de verificaciones en la función.

Paso 3: Interrupción del Contrato
Al ser establecidas las direcciones nulas, las funciones dependientes de estas direcciones en el contrato quedan inutilizables. Esto incluye operaciones esenciales como stake, disputas y la interacción con otros contratos, paralizando todas las operaciones y lógicas de negocio vinculadas.

Paso 4: Consecuencias para los Usuarios
Los usuarios del contrato se encuentran con un sistema inoperante, incapaces de realizar o deshacer estacados, participar en ventas o resolver disputas. Esto no solo afecta su capacidad de operar dentro del ecosistema sino que también mina la confianza en la plataforma y puede conllevar a la pérdida de fondos o activos bloqueados en el contrato.

Explicación Mitigación: La mitigación requiere la implementación de verificaciones para asegurar que las nuevas direcciones de contrato no sean nulas antes de actualizarlas en updateConfig. Esto prevendría la configuración accidental o maliciosa de direcciones inválidas.

Código Corregido:

```
function updateConfig(address newDao, address newShop, address newNft) external onlyRole(ADMIN_ROLE) {
    require(newDao != address(0), "DAO address cannot be zero");
    require(newShop != address(0), "Shop address cannot be zero");
    require(newNft != address(0), "NFT address cannot be zero");

    daoContract = IFP_DAO(newDao);
    shopContract = IFP_Shop(newShop);
    nftContract = newNft;

    emit NewConfig(newDao, newShop, newNft);
}
```

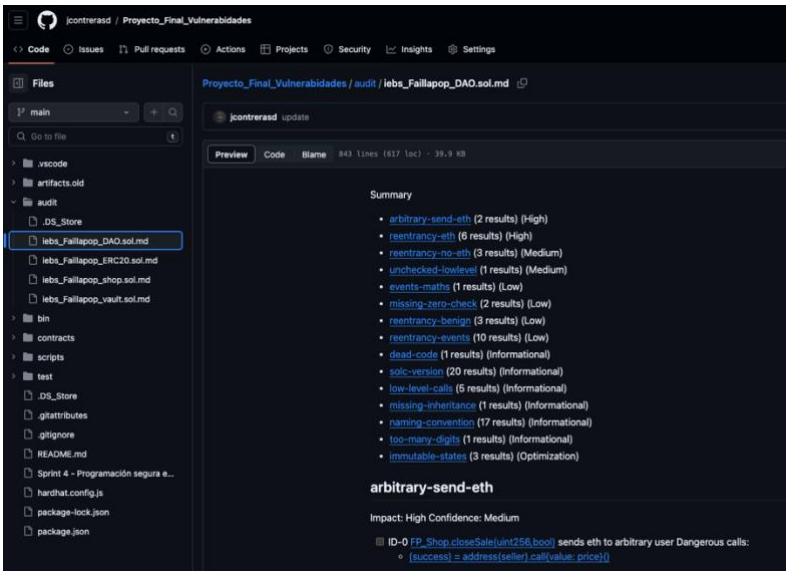
10. Funciones Adicionales de Seguridad

Funciones	Descripción	Beneficios
Emergency Stop (Pausa de Emergencia)	Implementación de un mecanismo que puede detener ciertas funcionalidades del contrato en caso de descubrirse una vulnerabilidad.	Permite una respuesta rápida para prevenir daños mayores mientras se investiga y corrige el problema.
Escape Hatch (Salida de Emergencia)	Permite retirar fondos a una dirección segura en caso de que el contrato se vea comprometido.	Protege los fondos de los usuarios permitiendo que sean recuperados en situaciones críticas.
Multisig	Requiere que múltiples partes autoricen una transacción antes de que pueda ejecutarse, a menudo implementado a través de un contrato de cartera multisig.	Aumenta la seguridad al requerir consenso para acciones críticas, reduciendo el riesgo de mal uso o robo por parte de un solo actor.
Rate Limiting (Limitación de Frecuencia)	Restricciones en la frecuencia de transacciones para prevenir abusos y reducir posibles daños en caso de ataque.	Limita la cantidad de fondos que podrían ser afectados en un periodo de tiempo determinado.
Timelocks	Introduce demoras obligatorias en la ejecución de ciertas funciones críticas.	Da tiempo para que las acciones sospechosas sean notadas y potencialmente revertidas antes de que causen daño.
Upgradability (Actualizaciones)	Permite actualizar el contrato para introducir mejoras o corregir fallos sin perder el estado o los fondos.	Asegura la capacidad de mejorar la seguridad con el tiempo sin tener que migrar recursos a un nuevo contrato.

11. Anexo de Revisión con Slither

slither . --checklist --show-ignored-findings --markdown-root ../ > ./audit/iebs_analisis_vulnerabilidades.md

Se creo un carpeta **audit** en **github** con la salida de este análisis, que se encuentra en formato markdown y contienen links hacia las lineas de los archivos .sol con los smart contract.



Importante : Considerar el siguiente comentario, dado que si bien la creación de cada archivo MD es independiente, se debe tener presente que los Smart Contract tienen referencias a otros Smart contract, lo que hace que el informe creado por slither debe filtrar por Smart Contract que se desea someter a análisis.

FAQ (https://github.com/crytic/slither?tab=readme-ov-file#faq)
How do I exclude mocks or tests?

View our documentation on path filtering.
How do I fix "unknown file" or compilation issues?

Because slither requires the solc AST, it must have all dependencies available. If a contract has dependencies, slither contract.sol will fail. Instead, use slither . in the parent directory of contracts/ (you should see contracts/ when you run ls). If you have a node_modules/ folder, it must be in the same directory as contracts/. To verify that this issue is related to slither, run the compilation command for the framework you are using e.g npx hardhat compile. That must work successfully; otherwise, slither's compilation engine, crytic-compile, cannot generate the AST.