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

Versión 1.0 – Clase 3 Proceso de una auditoria.

SPRINT SEMANAL 3

Federico Cambero Fenoy
IEBS Business School

Mayo de 2023

Tabla de Contenidos

1. Informe de Auditoría.

2. Licencia.

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 del Sprint1.
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 para esta auditoria está en el repositorio:

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

Las versiones usadas son:

VulnerableVault.sol
a37df57a665de3909ecc05d774fa8dc7b08ed0d5

VulnerableShop.sol
88c50522d2d02f67e032f9cd724884ff6d046032

VulnerableDAO.sol
f7dfb6da38897f27f45ce5c3261bff2d7586996e

VulnerableBank.sol
3afb465fdbe5e6036f7574499e8b0d32a807b119

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 de resultados.

Nº	Descripción	Nivel de riesgo	Estado	contrato
1	Función “deletesale” debe ser “internal”	Critico	Pendiente	VulnerableShop
2	Función “reimburse” hace transferencia cantidad a devolver en tokens.	Critico	Pendiente	VulnerableShop
3	Función “lotteryNFT” calculo número aleatorio para entregar NFT	Critico	Pendiente	VulnerableDAO
4	Owner del contrato ha incluido una contraseña como control de acceso	Alto	Pendiente	VulnerableDAO
5	Modifier “enoughStaked” no comprueba si la resta que realiza puede dar el resultado por debajo de cero.	Critico	Pendiente	VulnerableVault
6	No se comprueba la llamada al contrato externo “powerseller_nft”	Alto	Pendiente	VulverableVault
7	El modifier “onlyOwner” incluye control acceso a través de tx.origin.	Alto	Pendiente	VulnerableBank
8	El modifier “returnRewards” hace llamada externa vulnerable a reentrada	Critico	Pendiente	VulnerableBank

7. Detalle de resultados.

1. Función “deletesale” debe ser “internal”

Nivel de Riesgo: Critico.

Impacto: Contrato VulnerableShop.sol

La función “deleteSale” está pensada para ser “internal”, pero ha sido marcada como “public”.

Esto hará que, en lugar de ser llamada solo de manera controlada desde “removeMaliciousSale”, también pueda ser usada por sí misma.

- Como la función afectada recibe el argumento “itemID”, un atacante simplemente podría llamar a la función para borrar Sales de manera arbitraria. Esto causaría una denegación de servicio completa de la plataforma, ya que todas las ventas creadas podrían ser borradas.

Recomendación: Cambiar la visibilidad de la función “deleteSale” a “Internal”

Estado: Pendiente.

2. Función “reimburse” hace transferencia cantidad a devolver en tokens.

Nivel de Riesgo: Critico.

Impacto: Contrato VulnerableShop.sol

La función “reimburse” hace una transferencia de la cantidad a devolver en tokens. En lugar de dirigir la transferencia al usuario beneficiario de la devolución, la dirige a “msg.sender”.

- Cada vez que el owner realice una devolución, esa misma dirección recibirá el reintegro en lugar de su legítimo beneficiario.

Recomendación: Modificar la línea afectada de “reimburse” para que el destinatario de la transferencia sea “user” en lugar de msg.sender.

Estado: Pendiente.

3. Función “lotteryNFT” calculo número aleatorio para entregar NFT.

Nivel de Riesgo: Critico.

Impacto: Contrato VulnerableDAO.sol

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.

- 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”.

Recomendación: La función “LotteryNFT” debería usar un oráculo externo para obtener números aleatorios y realizar el sorteo.

Estado: Pendiente.

4. Owner del contrato ha incluido una contraseña como control de acceso.

Nivel de Riesgo: Alto.

Impacto: Contrato VulnerableDAO.sol

El "owner" de este contrato ha incluido una contraseña durante el despliegue para usarla como control de acceso. Como esta información está almacenada en el storage, el cual es público, cualquier podrá acceder a las funciones privilegiadas.

- Un atacante podría leer el storage del contrato de diferentes maneras para obtener el string de la contraseña. También se podría revisar el mensaje de despliegue e intentar obtener el valor entregado, pero es más difícil.

Recomendación: La autorización debe realizarse por whitelisting de direcciones en lugar de por contraseña.

Estado: Pendiente.

5. **Modifier “enoughStaked” no comprueba si la resta que realiza puede dar el resultado por debajo de cero.**

Nivel de Riesgo: Critico.

Impacto: Contrato VulnerableVault.sol

La función “unstake” comprueba correctamente que el usuario que llama tiene suficientes fondos no bloqueados para recuperar la cantidad deseada. Pero el modifier “enoughStaked” no comprueba si la resta que realiza puede dar un resultado por debajo de cero, como el contrato hace uso de solidity 0.7 hará underflow si la cantidad a reducir, efectivamente dando un número muy alto que será considerado válido. Después, la función “unstake” también resultará en un underflow al restar la cantidad a extraer, permitiendo a cualquier usuario malicioso vaciar el contrato.

- Sería posible llamar a unstake con balance cero y sacar cualquier cantidad de fondos.

Recomendación: El contrato debería usar solidity >= 8.0.0, la librería safeMath de Openzeppelin o reordenar la operación.

Estado: Pendiente.

6. No se comprueba la llamada al contrato externo “powerseller_nft”

Nivel de Riesgo: Alto.

Impacto: Contrato VulnerableVault.sol

La función “claimReward” comprueba que poseemos un NFT concreto haciendo una llamada al contrato externo “powerseller_nft” usando “call”. El resultado de esta llamada no es comprobado, con lo cual no hace efectos de control de acceso real.

> Un atacante sin privilegios podría llamar libremente a “claimRewards”, ya que nunca se sabría si es o no privilegiado.

Recomendación: La función “claimReward” debería comprobar el resultado de la llamada de bajo nivel y hacer un revert si ha fallado.

Estado: Pendiente.

7. El modifier “onlyOwner” incluye control acceso a través de tx.origin.

Nivel de Riesgo: Alto.

Impacto: Contrato VulnerableBank.sol

El modifier “onlyOwner” incluye un control de acceso que en lugar de comprobar msg.sender comprueba tx.origin. De esta manera, en lugar de comprobar quién manda el mensaje estará comprobando quién ha iniciado la transacción.

- Se podría orquestar un ataque de phishing contra el administrador, de forma que consiguieras que hiciera uso de un contrato malicioso intermedio. Este contrato malicioso podría llamar a la función “updateConfig” desde su código, suplantando al administrador y consiguiendo elegir libremente el periodo de distribución.

Recomendación: El modifier “onlyOwner” debe comprobar msg.sender.

Estado: Pendiente.

8. El modifier “returnRewards” hace llamada externa vulnerable a reentrada

Nivel de Riesgo: Critico

Impacto: Contrato VulnerableBank.sol

La función “distributeBenefits” aparentemente sigue el patrón CEI, pero el modificador “returnRewards” hace una llamada externa. Esto hace que sea vulnerable a reentrada, ya que antes de que se ejecute la secuencia segura de “doInvest”, “returnRewards” podrá ejecutar código externo sin que la variable de estado “total_vested” haya sido actualizada.

- Un atacante podría desplegar un smart contract malicioso para, aprovechando el ataque anterior, llamar a “doInvest” haciendo que el modificador “returnRewards” ejecute la llamada. La función receive() a su vez volverá a llamar a “doInvest”, y así sucesivamente. Con esto el atacante conseguirá vaciar el contrato extrayendo en forma de rewards todos los fondos.

Recomendación: Poner el elemento “_;” al principio del modificar o añadir reentrancyGuard a la función. También se podría sugerir mover la lógica contenida en el modificador “returnRewards” a una función a parte que se ejecute siguiendo el patrón ECI.

Estado: Pendiente.