Título:

Informe de Análisis de Amenazas – KipuBankV3

Autora:

Nidia Karina Garzón Grajales

ETH-KIPU/ Módulo 5:

Preparación para Auditoría

Fecha:

19/11/2026

1. Breve descripción general de cómo funciona KipuBankV3

KipuBankV3 es un contrato inteligente que implementa un banco / bóveda multiactivo sobre Ethereum. Su objetivo es permitir que los usuarios depositen y retiren tanto ETH nativo como tokens ERC-20, aplicando límites de seguridad y un modelo simple de roles administrativos.

A nivel de modelo de datos, el contrato mantiene un balance interno por usuario y por token usando un mapping anidado:

- balances[usuario][token] guarda cuánto tiene cada usuario de cada activo.
- Para representar ETH uso la dirección especial address(0) (constante NATIVE).

Además, llevo un registro del Total Value Locked por token a través de totalDepositedPerToken[token], que suma todos los depósitos de ese activo dentro del contrato. Sobre ese TVL se apoyan los límites (“caps”) de seguridad:

- bankCapPerToken[token]: límite global de cuánto puede haber depositado de ese token en la bóveda.
- withdrawCapPerToken[token]: límite por transacción para retiros de ese token.
- En el caso de ETH, también existe bankCapUsdETH, que es un tope en USD sobre el TVL de ETH, calculado con un oráculo de precios de Chainlink.

Para el control de acceso, KipuBankV3 combina Ownable con AccessControl:

- owner es el dueño del contrato (desplegador) y puede cambiar caps, oráculo y parámetros críticos.
- BANK_ADMIN_ROLE permite que otros administradores de banco ejecuten las mismas acciones operativas que el owner.
- Solo estas identidades privilegiadas pueden, por ejemplo, actualizar límites o usar las funciones de rescate de fondos.

En cuanto a funcionalidades principales para los usuarios, el contrato ofrece:

A. Depósito de ETH

   i. A través de depositETH() o enviando ETH directamente al contrato (receive()), el usuario suma saldo en balances[msg.sender][NATIVE].

   ii. Antes de aceptar el depósito se verifican:

      - Cap global del token ETH en wei (bankCapPerToken[NATIVE]), si está configurado.
      - Cap en USD (bankCapUsdETH), usando el precio ETH/USD obtenido vía AggregatorV3Interface de Chainlink.

B. Retiro de ETH

   i. La función withdrawETH(amount) permite retirar parte del saldo interno.

   ii. Se comprueba que:

      - El monto no sea cero.
      - No supere el withdrawCapPerToken[NATIVE] si este está configurado.
      - El usuario tenga balance suficiente.
      - Luego se actualizan los balances internos y se envía ETH con un call protegido mediante _safeTransferETH.

C. Depósito de tokens ERC-20

   i. El usuario primero aprueba al contrato y luego llama depositToken(token, amount).

   ii. Se valida que token no sea la dirección nativa, que el monto sea > 0 y que no se sobrepase el bankCapPerToken[token] si existe.

   iii. El contrato usa SafeERC20.safeTransferFrom para manejar tokens estándar y no estándar, y después actualiza balances y TVL.

D. Retiro de tokens ERC-20

   i. La función withdrawToken(token, amount) permite retirar saldo de un token concreto.

   ii. De nuevo se aplican:

      - Límite por transacción (withdrawCapPerToken[token]), si está configurado.
      - Verificación de saldo interno suficiente.

   iii. Finalmente se actualiza el estado y se ejecuta safeTransfer hacia el usuario.

Además de las operaciones básicas de banco, KipuBankV3 incluye una función de swap interno entre tokens ERC-20, swapVaultTokens(tokenIn, tokenOut, amountIn, minAmountOut). Esta función:

- Trabaja solamente con tokens ERC-20 (no permite NATIVE).
- Descuenta amountIn del saldo interno del usuario en tokenIn.
- Calcula amountOut normalizando las diferencias de decimales entre tokenIn y tokenOut, simulando un cambio 1:1 estilo stablecoins.
- Comprueba que el banco tenga suficiente liquidez de tokenOut (totalDepositedPerToken[tokenOut]).

- Verifica también que el resultado respete un mínimo (minAmountOut) para dar una pequeña protección contra slippage.
- Actualiza balances y TVL internamente, sin mover tokens on-chain, y emite el evento Swapped.

*En el plano de seguridad técnica, el contrato aplica varios patrones recomendados:

- nonReentrant (heredado de ReentrancyGuard) en todas las funciones que modifican balances y hacen transferencias.
- Patrón CEI (Checks-Effects-Interactions): primero validar, luego actualizar estado y al final interactuar con ETH o tokens externos.
- Uso de custom errors para ahorrar gas (ZeroAmount, InsufficientBalance, BankCapReached, etc.).
- Emisión de eventos (Deposited, Withdrawn, Swapped, CapsUpdated, etc.) para poder auditar fácilmente la actividad en Etherscan.

En resumen, KipuBankV3 se comporta como una bóveda multi-activo con límites configurables, oráculo de precios para ETH, swap interno simple entre tokens ERC-20 y un modelo claro de roles administrativos, intentando equilibrar funcionalidad de banco con medidas básicas de seguridad para un entorno de aprendizaje Web3.

---

2. Evaluar la madurez del protocolo

2.1 Cobertura de pruebas

En esta versión de KipuBankV3 he intentado aplicar una mentalidad de pruebas más cercana a lo que se ve en una auditoría real.

Las pruebas se han centrado en tres niveles:

A. Pruebas manuales en Remix y Sepolia
    - Verifiqué depósitos y retiros de ETH usando dos cuentas distintas.

- Probé depósitos y retiros de dos tokens ERC-20 diferentes (por ejemplo, MockDAI y MockUSDC).
- Validé que los caps de depósito y de retiro se respetaran cuando se configuraban valores bajos.
- Probé el swap interno swapVaultTokens cambiando saldo de un token estable a otro, comprobando que los balances internos quedaran consistentes.

B. Pruebas unitarias con Foundry (forge test)

Preparé una primera batería de tests en Foundry para automatizar los casos básicos:

- Depósito y retiro exitoso de ETH.
- Depósito y retiro exitoso de un token ERC-20.
- Reversión cuando el usuario intenta retirar más de su saldo.
- Reversión cuando el retiro supera el withdrawCapPerToken.
- Prueba de que el swap descuenta tokenIn, acredita tokenOut y no rompe el TVL total.
- Estos tests no cubren todavía todos los escenarios extremos, pero sirven como base para seguir ampliando la cobertura.

C. Escenarios adversarios dirigidos

Basándome en las vulnerabilidades Open Worldwide Application Security Project (OWASP 2025), revisé manualmente:

- qué pasa si el owner pone caps muy pequeños o grandes,
- qué ocurre si un usuario intenta swappear sin liquidez suficiente en tokenOut,
- y cómo se comporta el contrato cuando el oráculo falla o devuelve un precio inválido.

En resumen, la cobertura actual es moderada: sigo dependiendo mucho de pruebas manuales, pero ya existe una base en Foundry que me permite repetir los casos críticos sin tener que probar todo a mano cada vez.

2.2 Métodos de prueba utilizados

Para diseñar los tests intenté aplicar la “mentalidad para pruebas” del módulo:

A. Perspectiva de arquitectura
Revisé primero las dependencias externas:

- el oráculo de Chainlink para ETH/USD;
- los contratos ERC-20 externos que pueden comportarse de forma extraña;
- y el uso de call para enviar ETH.
A partir de ahí definí escenarios donde estas integraciones podrían fallar o devolver datos incorrectos.

B. Perspectiva del exploiter

¿Al trata de entender como no debería funcionar esto?”
Esto me llevó a probar:

- retiros repetidos por encima de los límites,
- manipulación de caps por parte del owner,
- swaps cuando la bóveda casi no tiene liquidez de tokenOut,
- y el comportamiento frente a montos cero.

C. CEI (Checks–Effects–Interactions)

Todas las funciones sensibles siguen el patrón CEI:

- primero se hacen los checks, luego se actualiza el estado y al final se realizan las transferencias o llamadas externas.
- Parte de las pruebas consistió en confirmar que este orden se respetara en los casos importantes (depósitos, retiros y swaps).

D. ReentrancyGuard + revert esperados
Aunque el contrato usa ReentrancyGuard, también considero importante probar explícitamente las condiciones de revert:

- ZeroAmount,
- InsufficientBalance,
- BankCapReached,
- WithdrawLimitExceeded,

- InsufficientLiquidity en los swaps.
  Comprobar estas rutas de error ayuda a detectar errores de lógica y falta de validación de entradas.

Todavía no implementé fuzzing ni verificación formal, pero el diseño ya está pensado para que sea relativamente sencillo añadir estas técnicas más adelante sobre las funciones clave.

2.3 Roles, poderes y superficie de ataque

KipuBankV3 utiliza dos niveles de privilegios:

A. Owner (Ownable)
   - Puede cambiar caps de depósito y retiro para cualquier token.
   - Puede actualizar la dirección del oráculo Chainlink.
   - Puede cambiar el cap en USD para ETH.
   - Puede ejecutar las funciones de rescate de ETH y ERC-20.

B. BANK_ADMIN_ROLE (AccessControl)
   - Comparte prácticamente los mismos permisos operativos que el owner.
   - La intención es que en un futuro este rol pueda delegarse a un “equipo de operaciones” sin exponer la clave del owner.

Los usuarios normales solo pueden:

- depositar ETH o tokens,
- retirar sus propios saldos,
- y hacer swaps internos entre tokens que ya están depositados en la bóveda.

Desde el punto de vista de madurez, el control de acceso está definido pero es centralizado: el owner sigue siendo una figura muy poderosa, desde la cual se podría:

- dejar caps en cero,
- mover fondos con las funciones de rescate,
- o cambiar el oráculo por uno mal configurado.

Más adelante, en la sección de vulnerabilidades, analizo estos poderes administrativos como vectores de riesgo.

2.4 Documentación y seguridad por diseño

El contrato incluye comentarios extensos en inglés que explican:

- el propósito de cada variable de estado,
- qué representa NATIVE,
- cómo se interpretan las caps,
- y la intención de la función de swap interna.

Además:

- se utilizan custom errors descriptivos en lugar de strings largos,
- se emitieron eventos claros (Deposited, Withdrawn, Swapped, CapsUpdated, etc.) para facilitar el monitoreo,
- se aplicó el modificador nonReentrant en todas las funciones que combinan actualización de balances y transferencias.

Aun así, el protocolo no está todavía al nivel de “producción real”:
faltan especificaciones formales de comportamiento, una suite de tests más amplia, fuzzing automático y una auditoría externa independiente.
Por ahora lo considero un contrato en etapa de aprendizaje avanzado, con una base razonable pero todavía lejos de un estándar profesional.

---

3. Vectores de ataque y modelo de amenazas

En esta sección analizo KipuBankV3 desde la mentalidad de pruebas del módulo: arquitectura, invariantes (a nivel intuitivo) y, sobre todo, la perspectiva del exploiter (“¿cómo NO debería funcionar esto?”).

No encontré vulnerabilidades críticas de reentrancy gracias al uso de ReentrancyGuard y al patrón CEI, pero sí identifiqué varios puntos donde el protocolo puede ser atacado o quedar en un estado inseguro, especialmente en:

- el diseño económico del swap,
- la compatibilidad con ciertos tipos de tokens ERC-20,
- y los poderes administrativos del owner / bank admin.

A continuación, describo 4 escenarios de ataque, mapeados a las categorías OWASP 2025, con su criticidad y mitigación propuesta.

3.1. Vulnerabilidad 1 – Swap 1:1 sin fuente de precio (riesgo económico grave)

- Tipo OWASP principal
  SC02: Manipulación de Oráculos de Precio / SC03: Errores de Lógica.
- Criticidad: Alta

Descripción

La función:

function swapVaultTokens(

address tokenIn,

address tokenOut,

uint256 amountIn,

uint256 minAmountOut

) external nonReentrant

permite a un usuario cambiar saldo interno de tokenIn por saldo de tokenOut.
El cálculo de amountOut es:

uint8 decimalsIn = IERC20Metadata(tokenIn).decimals();

uint8 decimalsOut = IERC20Metadata(tokenOut).decimals();

uint256 amountOut = normalizeDecimals(amountIn, decimalsIn, decimalsOut);

Es decir, el contrato asume un tipo de cambio 1:1, ajustando solo por diferencias de decimales entre tokens. No se consulta ningún oráculo de precios ni hay una curva AMM (tipo x·y = k) que refleje oferta y demanda.

Esto funciona razonablemente si ambos tokens son stablecoins 1:1 (MockDAI vs MockUSDC), pero se convierte en un problema cuando:

- los tokens tienen valores de mercado muy distintos,
- o uno de los tokens prácticamente no tiene valor.

Escenario de explotación

Un atacante podría:

A. Identificar dos tokens permitidos por el protocolo, por ejemplo, tokenBarato y tokenCaro.
B. Depositar una gran cantidad de tokenBarato en la bóveda.
C. Llamar a swapVaultTokens(tokenBarato, tokenCaro, amountIn, minAmountOut) con un minAmountOut muy bajo.

D. El contrato le acreditará prácticamente la misma cantidad de tokenCaro (solo ajustando decimales), aunque su valor real en el mercado sea mucho mayor.

E. Finalmente, el atacante puede retirar tokenCaro con withdrawToken y venderlo fuera del contrato.

En la práctica, esto equivale a drenar el token más valioso depositado por otros usuarios, usando el vault como si fuera un “cajero” que regala el activo caro a cambio de un activo casi sin valor.

Impacto

- Pérdida directa de fondos para el protocolo y, por extensión, para los depositantes del token más valioso.
- Desbalanceo completo de la liquidez interna.
- Posible colapso económico de la bóveda, ya que el activo “caro” se vacía y solo quedan tokens poco valiosos.

Justificación de criticidad

Lo considero Alta porque:

- permite un ataque directo y repetible,
- no requiere romper permisos ni depender de fallos externos,
- y afecta a la integridad económica del sistema (los usuarios pierden valor real).

Mitigación propuesta

- Limitar la función swapVaultTokens exclusivamente a pares de stablecoins whitelisteados que mantengan paridad fuerte 1:1.
- Alternativamente, eliminar el swap en un entorno de producción y dejarlo solo para fines educativos.
- Si se quiere mantener un swap realista, integrarlo con:
    - un oráculo de precios por par,
    - o una curva de liquidez tipo AMM,
      y tests exhaustivos que cubran escenarios de desequilibrio.

3.2. Vulnerabilidad 2 – Supuestos sobre ERC-20 (tokens con fee o comportamiento no estándar)

A. Tipo OWASP principal:
SC03: Errores de Lógica / SC06: Llamadas Externas No Verificadas.

B. Criticidad: Media–Alta

Descripción

En depositToken el contrato hace:

IERC20(token).safeTransferFrom(msg.sender, address(this), amount);

balances[msg.sender][token] += amount;

totalDepositedPerToken[token] += amount;

Se asume que el contrato va a recibir exactamente amount unidades del token.
Sin embargo, muchos tokens “de la vida real” pueden:

- cobrar una comisión al transferir (fee-on-transfer),
- quemar parte del monto,
- o aplicar lógica especial que hace que el contrato reciba menos tokens de los que el usuario cree que envía.

En esos casos, el estado interno del vault quedaría desincronizado:

- El usuario tendría acreditado amount en balances[msg.sender][token],
- Pero el contrato realmente tendría menos de amount unidades.

Más adelante, cuando el usuario intente retirar:

IERC20(token).safeTransfer(msg.sender, amount);

la transferencia puede revertir por falta de balance en el contrato, bloqueando la retirada.

Escenario de explotación

No es necesario un atacante malicioso complejo; basta con que:

A. El protocolo permita depositar un token con fee-on-transfer o con lógica de quema.

B. Varios usuarios depositen ese token.

C. Al intentar retirar, el contrato no tenga suficiente balance real para cubrir todos los saldos internos.

El resultado práctico es un DoS parcial para ese token:

los usuarios no pueden sacar todo lo que “dice” su balance.

Impacto

- Los usuarios pueden quedar con fondos atrapados en el contrato para ese token concreto.
- Se pierde confianza en la bóveda.
- Se complica mucho el proceso de migración o reparación.

Justificación de criticidad

Lo clasifico como Media–Alta porque:

- no siempre se manifestará (solo con ciertos ERC-20),
- pero cuando ocurre afecta a la integridad de saldos y a la capacidad de retiro, que es una propiedad fundamental del protocolo.

Mitigación propuesta

- Mantener una lista blanca de tokens soportados explícitamente (por ejemplo, solo stablecoins estándar bien conocidas).
- Añadir tests específicos con tokens simulados que cobren fees para detectar este tipo de desajustes.
- En caso de querer soportar tokens con fee, ajustar la lógica de depósito para leer el balance del contrato antes y después y acreditar solo la diferencia real.

3.3. Vulnerabilidad 3 – Poderes administrativos y funciones de rescate sin límites

A. Tipo OWASP principal:
SC01: Vulnerabilidades de Control de Acceso / SC10: Ataques de Denegación de Servicio (DoS).
B. Criticidad: Alta

Descripción

El contrato define un modificador:

```
modifier onlyOwnerOrAdmin() {
    require(
```

```
        owner() == msg.sender || hasRole(BANK_ADMIN_ROLE, msg.sender),
        "Access denied: must be owner or bank admin"
    );
    _;
}
```

Este modificador protege, entre otras, las funciones:

function setCapsForToken(...)

function setOracle(...)

function setBankCapUsdETH(...)

function rescueERC20(address token, uint256 amount, address to)

function rescueETH(uint256 amount, address to)

En particular, rescueERC20 y rescueETH permiten mover cualquier cantidad de fondos del contrato hacia una dirección arbitraria, sin comprobar si esos fondos están asociados a depósitos de usuarios.

En la práctica, esto significa que:

- el owner o cualquier cuenta con BANK_ADMIN_ROLE tiene capacidad de vaciar por completo la bóveda,
- ya sea de forma intencional, por error, o si su clave es comprometida por un atacante.

Además, a través de setCapsForToken y setBankCapUsdETH, estos mismos roles pueden:

- establecer caps tan bajos que ya no se puedan hacer depósitos ni retiros prácticos (DoS de la funcionalidad),
- o desactivar los límites poniendo las caps en cero (0 = sin límite), aumentando la superficie de riesgo económico.

Escenario de explotación

A. Un atacante roba la clave privada de un admin o del owner.

B. Llama a rescueERC20(token, balanceTotal, addressDelAtacante) y rescueETH(...).

C. Todos los fondos quedan transferidos fuera del contrato, sin que los usuarios puedan hacer nada.

Incluso sin atacante, un error de operación (por ejemplo, rescatar tokens pensando que son “sobrantes”) puede dejar al protocolo insolvente frente a sus depositantes.

Impacto

- Pérdida total o parcial de los fondos depositados.
- Bóveda inutilizable hasta que se reponga liquidez.
- Riesgo reputacional muy alto.

Justificación de criticidad

La considero Alta porque:

- el vector depende solo de un rol privilegiado (owner / admin),
- y el impacto es máximo: pérdida de los fondos de todos los usuarios.

Es una decisión de diseño (como muchos protocolos centralizados al inicio), pero desde el punto de vista de amenazas debe quedar explícitamente reconocida.

Mitigación propuesta

- Limitar rescueERC20 y rescueETH para que solo puedan extraer saldos que no estén asociados a balances de usuarios, calculando la diferencia entre el balance real del contrato y totalDepositedPerToken.
- Añadir un time-lock o retraso para operaciones críticas, de forma que la comunidad tenga tiempo de reaccionar.
- Utilizar un multising en lugar de una sola clave para owner y BANK_ADMIN_ROLE.
- Documentar claramente este riesgo para que los usuarios entiendan el nivel de confianza que se deposita en la administración del protocolo.

3.4. Vulnerabilidad 4 – Configuración extrema de caps (0 o valores muy bajos)

- Tipo OWASP principal:
  SC03: Errores de Lógica / SC10: DoS.
- Criticidad: Media

Descripción

Los límites (caps) se interpretan así en el contrato:

// 0 = no cap

mapping(address => uint256) public bankCapPerToken;

mapping(address => uint256) public withdrawCapPerToken;

uint256 public bankCapUsdETH; // 0 = disabled

Es decir:

- bankCapPerToken[token] = 0 → depósitos ilimitados para ese token.
- withdrawCapPerToken[token] = 0 → sin límite por transacción en retiros.
- bankCapUsdETH = 0 → sin tope en USD para el TVL de ETH.

Por otro lado, nada impide que el owner/admin configure caps extremadamente bajos, como:

- un withdraw cap de 1 wei,
- o un bank cap prácticamente igual al TVL actual, impidiendo nuevos depósitos.

Escenarios de riesgo

1. DoS por caps muy bajos
    - Si withdrawCapPerToken[token] se fija en un valor extremadamente pequeño por error, los usuarios podrían retirar solo cantidades ridículas, haciendo el protocolo prácticamente inutilizable en la práctica.
2. Ausencia de límites con caps en cero
    - Si se ponen todos los caps en 0, el sistema se queda sin ningún freno para:
        - el crecimiento del TVL,
        - el tamaño de los retiros individuales,
        - y la exposición a movimientos bruscos de mercado.
    - En la práctica, esto no es un bug inmediato de código, pero sí una configuración de riesgo elevado.
3. Combinación con la vulnerabilidad de administración
    - Si un admin malicioso sube los caps o los pone en cero y luego utiliza funciones de rescate o estrategias económicas (por ejemplo, el swap 1:1), el impacto de los ataques anteriores se amplifica.

Impacto

- El protocolo puede quedar bloqueado en la práctica (DoS suave) o, en el otro extremo, operar sin ningún límite razonable de riesgo.

- Depende fuertemente de cómo configuren estos valores las personas con permisos de administración.

Justificación de criticidad

Lo califico como Media porque:

- por sí misma no rompe el contrato ni genera un exploit automático,
- pero expone al protocolo a errores humanos u operaciones maliciosas que pueden generar DoS o pérdidas económicas.

Mitigación propuesta

- Definir rangos razonables para los caps (mínimos y máximos) y validarlos con require.
- Establecer valores por defecto seguros y documentados.
- Incluir tests específicos donde se pruebe el comportamiento del protocolo con caps extremos.
- Revisar periódicamente la configuración en una checklist de operación segura.

En este cuadro hice el consolidado de la severidad de estas vulnerabilidades y aunque no rompen el contrato de inmediato, sí pueden comprometer los fondos si el protocolo se usa de forma real. Por eso las clasifiqué en severidades Media y Alta de acuerdo con lo observado en clase.

ID	Vulnerabilidad	Probabilidad	Severidad
3.1	Swap 1:1 sin fuente de precio	Excepcional	Alta
3.2	Supuestos sobre ERC-20	Rara	Media - Alta
3.3	Poderes administrativos y funciones de rescate sin límites	Excepcional	Alta
3.4	Configuración extrema de caps (0 o valores muy bajos)	Ocasional	Media

---

4. Métodos de prueba y cobertura (Foundry)

Para complementar las pruebas manuales realizadas en Remix y Sepolia, preparé un conjunto de pruebas automatizadas usando Foundry. El objetivo principal no fue alcanzar el 100% de cobertura, sino validar que las funciones críticas del protocolo:

- aceptan entradas válidas,
- rechazan entradas incorrectas,
- actualizan el estado interno correctamente,
- y cumplen las verificaciones de seguridad implementadas (caps, saldos, límites por transacción y reentrancy).

Para esta etapa empleé los comandos principales:

- forge build
- forge test -vvv
- forge coverage

La opción -vvv permite ver los revert codes, el orden de ejecución y el detalle de cada prueba, lo que es útil para detectar lógica incorrecta o errores silenciosos.

El comando forge coverage generó un reporte visual que muestra qué funciones fueron ejecutadas y qué líneas quedaron sin cubrir.

Preparé pruebas para:

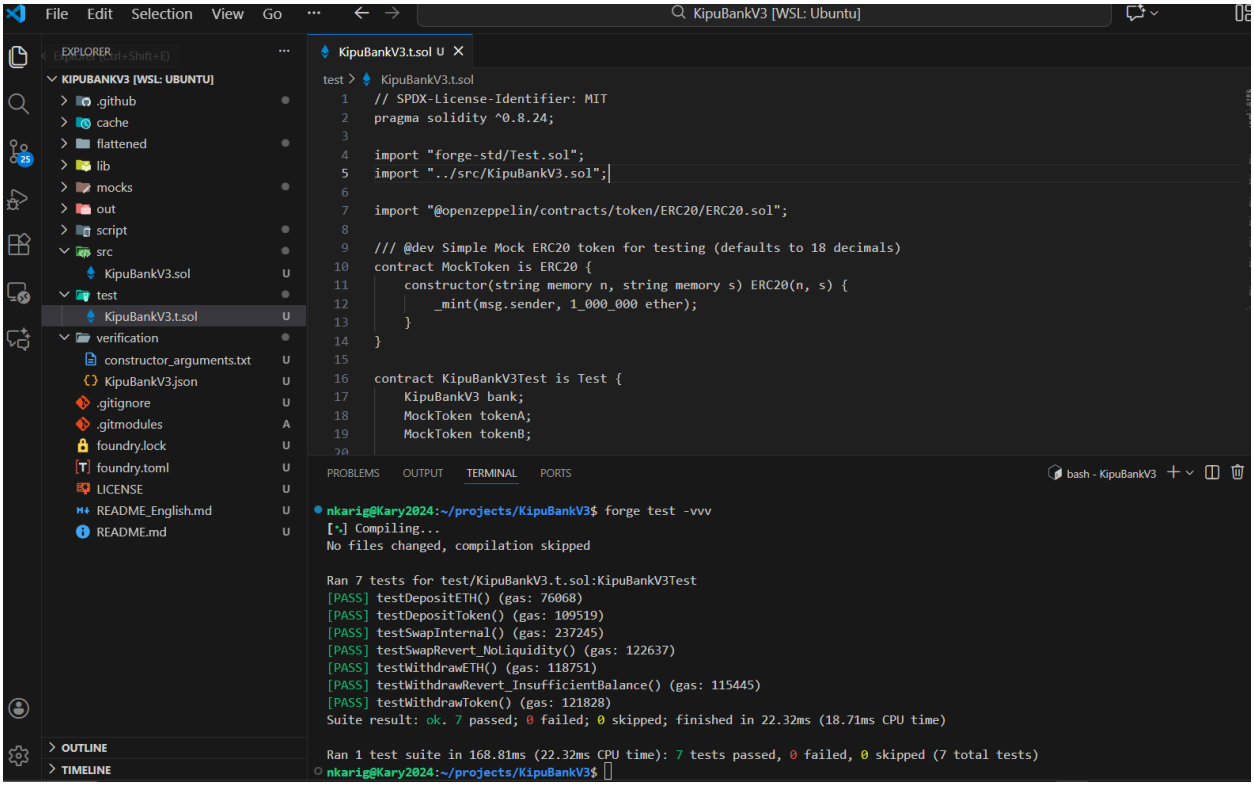
- Depósito y retiro de ETH
- Depósito y retiro de un ERC-20 estándar
- Reversión al intentar retirar más del saldo disponible
- Reversión al superar el límite withdrawCapPerToken
- Swap interno ERC20 ↔ ERC20 con liquidez suficiente
- Reversión cuando no hay liquidez suficiente para el swap

Estas pruebas confirmaron que el contrato:

- Respeta el patrón **Checks–Effects–Interactions**
- Utiliza **ReentrancyGuard** correctamente
- Mantiene la coherencia del estado interno tras operaciones consecutivas

A continuación se incluyen capturas de pantalla del reporte de cobertura como evidencia del proceso y análisis desde la terminal.

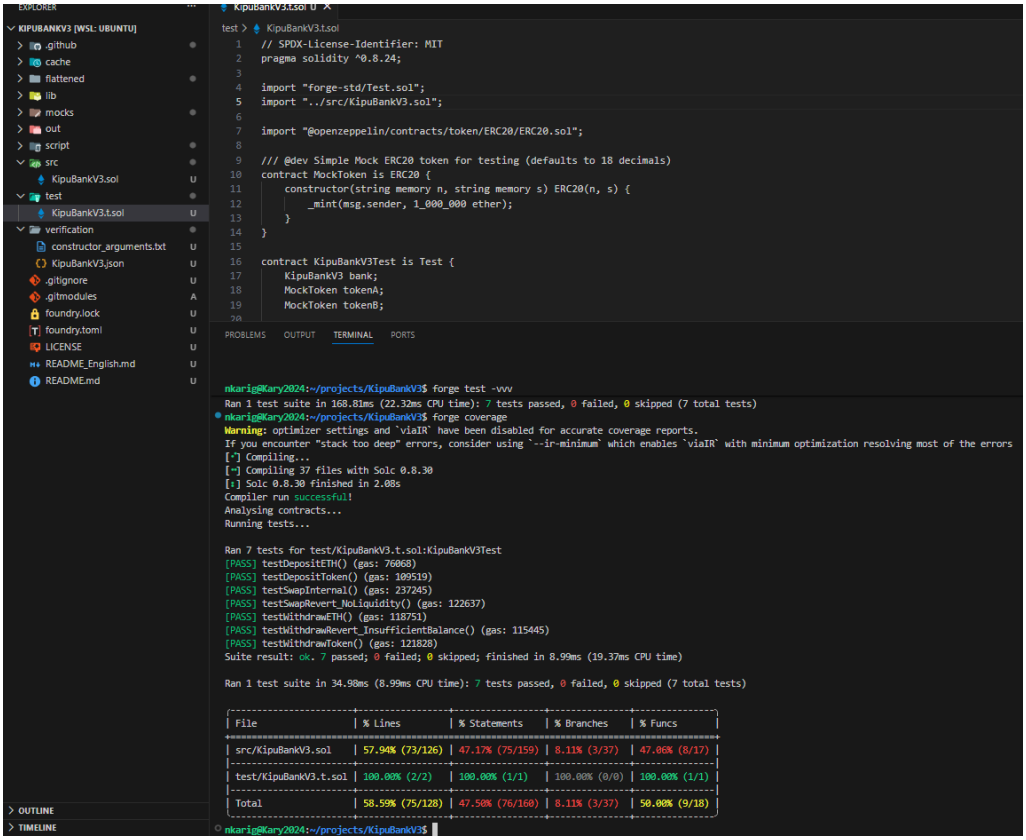
1. Forge test -vvv



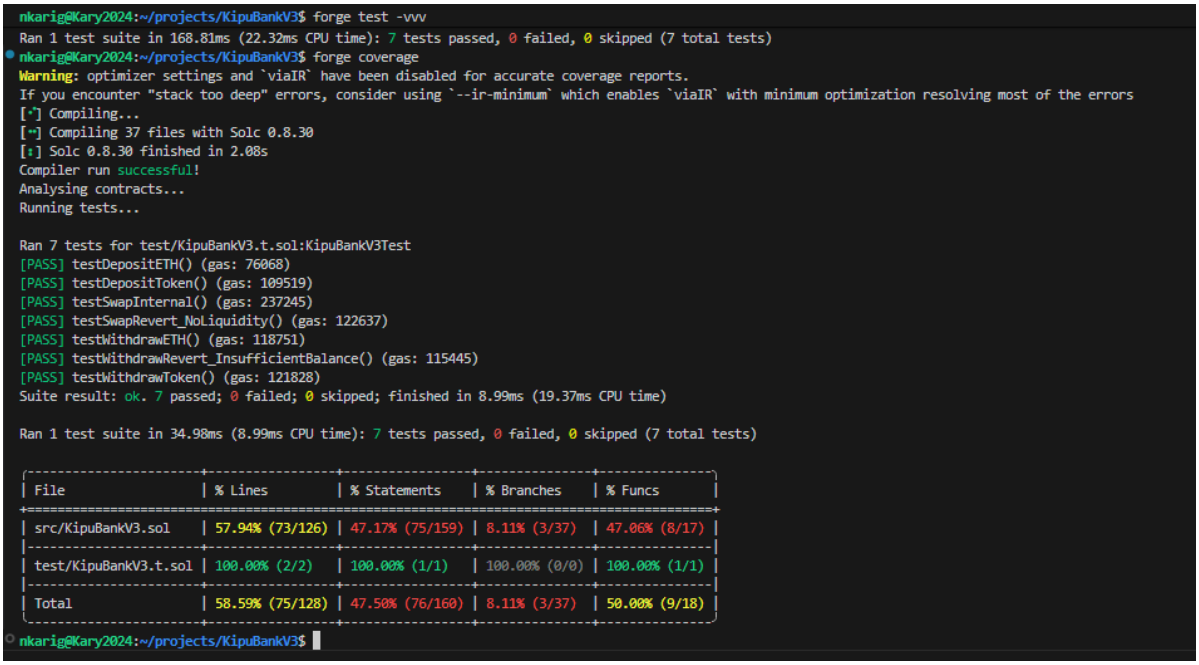
1.1 Forge test -vvv

```
nkarig@Kary2024:~/projects/KipuBankV3$ forge test -vvv
[⠊] Compiling...
No files changed, compilation skipped

Ran 7 tests for test/KipuBankV3.t.sol:KipuBankV3Test
[PASS] testDepositETH() (gas: 76068)
[PASS] testDepositToken() (gas: 109519)
[PASS] testSwapInternal() (gas: 237245)
[PASS] testSwapRevert_NoLiquidity() (gas: 122637)
[PASS] testWithdrawETH() (gas: 118751)
[PASS] testWithdrawRevert_InsufficientBalance() (gas: 115445)
[PASS] testWithdrawToken() (gas: 121828)
Suite result: ok. 7 passed; 0 failed; 0 skipped; finished in 22.32ms (18.71ms CPU time)

Ran 1 test suite in 168.81ms (22.32ms CPU time): 7 tests passed, 0 failed, 0 skipped (7 total tests)
nkarig@Kary2024:~/projects/KipuBankV3$
```

2. Forge coverage



2.1 Forge coverage



- En la ejecución de forge coverage para esta versión del contrato se obtuvo aproximadamente un 57.94 % de líneas cubiertas y un 47.17 % de statements en src/KipuBankV3.sol. Aunque no es una cobertura completa, sí confirma que los flujos principales (depósitos, retiros, swaps y reverts básicos) están siendo ejercitados de forma sistemática.

---

5. Recomendaciones

Después de revisar a fondo cómo funciona KipuBankV3, cómo se comporta con distintos escenarios y qué tan fácil sería romperlo si alguien lo intenta explotar, estas son las mejoras que considero necesarias para fortalecerlo. No son cambios obligatorios, pero sí son cosas que yo, como desarrolladora, tendría en cuenta para una versión más madura.

5.1 Mejorar el sistema de swap interno

- El swap actual funciona, pero es muy básico: cambia 1:1 sin ver precios reales. Para fines educativos está perfecto, pero para un sistema real no sería seguro.
- Por eso recomiendo:
- **Usar solo stablecoins whitelisteadas,** y dejar claro que fuera de ese caso el swap puede ser peligroso.
- Documentar el mecanismo como "didáctico" para evitar confusiones.

- A futuro, si quisiera un swap real, tocaría:
    - integrar oráculos de precio por par,
    - o directamente aplicar una curva AMM tipo Uniswap.

No necesito que esto exista ahora, pero sí dejar claro que el diseño actual tiene límites.

5.2 Controlar mejor los tokens permitidos

Muchos ERC-20 allá afuera se comportan raro (tienen fees, queman tokens, hacen rebases). KipuBankV3 asume que todos los tokens se comportan estándar, y ahí puede haber problemas.

Recomendaciones:

- Mantener una **lista blanca** de tokens soportados.
- Crear pruebas con “tokens de mentira” que cobren fees para ver si el protocolo se desincroniza.
- Si algún día quisiera soportar tokens con fee, tendría que ajustar la lógica de depósito para acreditar lo que realmente entró al contrato, no lo que el usuario envió.

5.3 Reducir riesgos asociados al owner / bank admin

Algo que aprendí en este módulo es que el rol del owner puede ser un arma de doble filo.
En este contrato el owner y los admins pueden prácticamente hacer todo, incluso rescatar fondos sin restricciones. Eso no es un bug, pero sí es una responsabilidad grande.

Mis recomendaciones personales:

- Agregar reglas para que rescueETH y rescueERC20 solo puedan mover fondos que no pertenecen a los usuarios.
- Usar un multisig para owner y admin (menos riesgo de hackeo).
- Agregar un time-lock para operaciones sensibles.

5.4 Validar bien los caps para evitar configuraciones peligrosas

Los caps permiten controlar cuánto entra y cuánto se retira, pero como el valor “0” significa “sin límite”, eso puede ser riesgoso.

Para evitar problemas recomiendo:

- Definir rangos razonables (mínimos y máximos) y rechazarlos si son extremos.

- Documentar cuál es la configuración segura por defecto.
- Probar escenarios de caps muy bajos, caps enormes, caps en cero, etc.

5.5 Mejorar la suite de pruebas

Hice pruebas manuales y automáticas, pero todavía hay espacio para crecer.
Mis recomendaciones:

- Añadir fuzzing para ver combinaciones raras de inputs.
- Probar qué pasa cuando el oráculo falla o da precios inválidos.
- Probar todas las funciones administrativas.
- Crear tests específicos para caps extremos y swaps sin liquidez.

5.6 Documentación más clara y orientada a riesgos

Finalmente, recomiendo:

- Dejar en el README una explicación honesta del modelo de confianza del protocolo.
- Señalar los riesgos conocidos (swap 1:1, poderes del owner, tokens con fee).
- Aclarar que esta es una versión educativa, no un producto financiero real.

---

6. Conclusiones y próximos pasos

KipuBankV3 es la versión más completa que he construido hasta ahora dentro del módulo, y definitivamente fue la que más me hizo pensar en seguridad. A nivel de funcionalidad, el contrato cumple lo que promete: manejar depósitos de ETH y ERC-20, retiros, límites configurables y un swap interno simple. Además, respeta CEI, usa ReentrancyGuard, custom errors y roles claros.

Las pruebas manuales y las pruebas en Foundry me ayudaron a confirmar que:

- las funciones centrales se comportan como deberían,
- las rutas de error responden bien,
- y no encontré reentrancy gracias al diseño actual.

Cuando corrí forge coverage, la cobertura que obtuve fue aproximadamente:

- **57.94% de líneas cubiertas**

- **alrededor del 50% de funciones ejercitadas**

No es un porcentaje alto, pero sí suficiente para validar los flujos importantes. Ampliarlo dependería del tiempo y de si quiero fortalecer más la parte de testing.

6.1 Lo que realmente aprendí de este análisis

- El mayor riesgo del protocolo no es un bug técnico, sino económico: el swap 1:1.
- El segundo riesgo viene de tokens ERC-20 no estándar, que pueden romper los saldos internos.
- Y el tercer riesgo viene de la centralización en el **owner/admin,** que podría mover fondos por error o si su clave es comprometida.

Reconocer esto me ayuda a ver el contrato con más objetividad.

6.2 Mis próximos pasos si quisiera llevar este proyecto más lejos

a) Implementar las mitigaciones que propuse (swap, rescates, caps).
b) Añadir fuzzing y ampliar varias pruebas adversarias.
c) Cambiar roles a multisig.
d) Construir una versión “KipuBankV4” con swap seguro o eliminar el swap.
e) Buscar una revisión externa para ver si otras personas encuentran cosas que yo no.

6.3 Cierre personal

KipuBankV3 es un proyecto que me permitió ver, de primera mano, cómo una funcionalidad que parece sencilla (como un swap 1:1 o un cap mal configurado) puede convertirse en un riesgo real.

Después de este análisis, siento que tengo una visión más completa de lo que significa mirar un contrato con ojos de seguridad, no solo con ojos de desarrolladora. Y eso, más allá del código, fue lo más valioso del módulo.