BVCC Wallet

BLOCK VENTURE CHAIN CAPITAL

Agent Wallet Informe de Seguridad y Pruebas

Análisis estático, remediación de vulnerabilidad y batería de pruebas de permisos del agente on-chain

OBJETO	BVCC Agent Wallet (smart account ERC-4337 / ERC-7579)
CÓDIGO	Commit `a3e28f1` · árbol de `contracts/src` `337e84f0` · generación V4
DESPLIEGUE	Arbitrum One · Ethereum · BNB Chain · Base · Polygon · Arbitrum Sepolia
TRABAJO	Revisión de seguridad **interna** y verificación de la remediación — no es una auditoría externa
PERIODO	2026-06-06 → 2026-07-29 · cuatro rondas de revisión
HALLAZGOS	2 Críticos · 3 Altos · 5 Medios · 1 Bajo · 2 Informativos
ESTADO	9 remediados · 4 abiertos o aceptados · 303/303 tests

Índice

Los hallazgos se citan por identificador (`BVCC-01`) a lo largo de todo el documento, no por número de sección. Las secciones van por número y los anexos por letra.

1 Resumen ejecutivo

La BVCC Agent Wallet es una cuenta inteligente ERC-4337 que permite a un dueño delegar autoridad de gasto acotada en un agente autónomo (una EOA). Este informe cubre cuatro rondas de revisión de esa vía de delegación, desde la primera batería de permisos en junio de 2026 hasta la generación V4 desplegada el 2026-07-28.

13	2	9	303
HALLAZGOS, CÓDIGO ACTUAL	CRÍTICOS	REMEDIADOS	TESTS, TODOS PASAN

Qué encontró la revisión

Dos hallazgos son Críticos. **BVCC-01**, una reentrada cruzada, permitía que un agente cuya dirección adquiriera código después reentrara al `execute()` de nivel dueño y se saltara todos los límites configurados de una vez; una prueba de concepto movió 49,925 ETH contra un presupuesto vitalicio de 0,003 ETH, y el mismo ataque funcionó contra una wallet creada desde la factory V2 viva sobre un fork de mainnet. **BVCC-02**, una máscara de comandos equivocada en el validator del Universal Router, habría aceptado un depósito en un puente cross-chain como si fuera un swap —justo la clase de exfiltración que el validator existe para impedir— y se cazó en revisión antes de que ese validator llegara a registrarse en ninguna red.

Tres hallazgos son Altos: los presupuestos del agente no acotan el gasto en protocolos cuando existe una allowance de token viva (**BVCC-03**, **abierto**), los guardianes los podía elegir quien llamara primero a la factory abierta (**BVCC-04**), y una recuperación completada no detenía a los agentes autorizados antes de ella (**BVCC-05**). El resto son Medios o menores.

Qué se hizo al respecto

Nueve hallazgos están remediados y vivos. Siete de ellos exigían cambiar el bytecode y viajaron juntos como la generación **V4**, desplegada y verificada de forma independiente en las seis redes soportadas el 2026-07-28. La validación profunda —la capa que inspecciona calldata que un anclaje de bitmap no sabe expresar— está completa: los dos selectores que delegan en el registry on-chain tienen validator activo en todas las redes, y el ciclo completo de liquidez de Uniswap v4 lo ejercitó un agente en mainnet de Arbitrum, terminando con la posición quemada y sin valor nativo varado.

Cuatro hallazgos siguen abiertos. Uno es Alto y es el riesgo residual material: donde el dueño ya ha concedido una allowance de token a un protocolo, anclar el destino de una llamada no acota su valor, así que una clave de agente comprometida vale más que su presupuesto. El plan escalonado está en **BVCC-03** y la mitigación operativa en §6. Los otros tres son Medios o Informativos y son limitaciones aceptadas y documentadas.

> **Dos salvedades acompañan a cualquier afirmación de que V4 arregló esto.** Primera: una cuenta inteligente desplegada no se puede actualizar en su sitio, porque su dirección se deriva de su bytecode. **Las wallets que sigan en V1, V2 o V3 corren el bytecode vulnerable hasta que sus dueños migren**, y BVCC-01 afectaba a las tres generaciones por igual. Segunda: BVCC-03 sobrevive a V4 y no es asunto de contratos — pon a cero cualquier allowance viva hacia un protocolo antes de autorizar a un agente que pueda alcanzarlo.

Esta es una revisión interna. La hizo el equipo de desarrollo sobre su propio código, no un tercero independiente, y los contratos no han sido auditados externamente. Cuatro de los hallazgos —BVCC-04, BVCC-06, BVCC-07 y BVCC-11— salieron de una pasada posterior sobre el mismo código con una herramienta de análisis distinta, después de que tres pasadas anteriores no los vieran. Aquella pasada también fue interna: **ningún tercero independiente ha revisado este código.** El §7 expone los límites de lo que este documento establece.

2 Alcance, metodología y modelo de severidad

Código revisado

ELEMENTO	VALOR
Repositorio	`blockventurechaincapital-crypto/bvcc-wallet-agent`
Commit	`a3e28f1a7c8e265d5b3630f98e4c8e77c6752eb5`
Árbol de `contracts/src`	`337e84f05fa11b948e3ad3221398ea4edc030f45`
Build	solc 0.8.36 · `via_ir` · `optimizer_runs = 50` · EVM `cancun` (congelada — las direcciones CREATE2 dependen de ella)
Herramientas	Slither 0.11.3 · Foundry (forge 1.5.1) · viem 2.x

Las rondas anteriores revisaron los árboles de V1, V2 y V3 en sus propios commits; esos hallazgos y sus remediaciones están en el Anexo D y el Anexo E.

Dentro del alcance

- `BVCCAgentWallet.sol` — lógica de permisos del agente (`executeAsAgent`, topes por token y de ETH, pausa, caducidad, la capa de call policies).
- `BVCCWallet.sol` — cuenta inteligente base (ejecución, comisiones, recuperación por guardianes).
- `BVCCAgentWalletFactory.sol` / `BVCCWalletFactory.sol` — factories CREATE2.
- `BVCCValidatorRegistry.sol`, `BVCCUniversalRouterValidator.sol`, `BVCCPositionManagerValidator.sol`, `BVCCHookRegistry.sol` y la interfaz `IBVCCValidator`.
- Los presets de call policy que registra la interfaz, en tanto que configuración que determina el comportamiento on-chain.

Fuera del alcance

- Los protocolos de terceros con los que interactúa la wallet (Uniswap, Aave, Permit2): se asume su propia seguridad, y la wallet se revisa partiendo de que pueden comportarse de forma adversarial en su interfaz.
- Infraestructura de bundler, paymaster y EntryPoint.
- La interfaz web, salvo donde registra policy on-chain o es el único sitio donde existe un control (se señala explícitamente cuando es el caso).
- Riesgo económico y de mercado, incluida la manipulación de precios en los pools contra los que opera la wallet, salvo donde es el mecanismo de un hallazgo (BVCC-03).

Metodología

- **Revisión manual** de la vía de ejecución del agente, la contabilidad de permisos, la recuperación y la capa de call policies, en cuatro pasadas a lo largo de las cuatro generaciones.
- **Análisis estático** — Slither 0.11.3 sobre el árbol en cada ronda; resultados y registro de falsos positivos en el Anexo C.
- **Pruebas unitarias, de fork y de fuzz** — Foundry. Cada hallazgo tiene su test de regresión. El validator del router se fuzzea sobre todo su espacio de comandos contra un modelo del router

real; Aave se ejercita sobre un fork de mainnet.
- **Desarrollo de exploits** — los hallazgos se confirman con una prueba de concepto que funciona, no con un argumento. BVCC-01 se reprodujo contra una wallet creada desde la factory V2 *viva* sobre un fork.
- **Pruebas on-chain** — la EOA del agente ejecuta `executeAsAgent` directamente. Los reverts se confirman por simulación (`eth_call`) contra estado vivo; los casos válidos, como transacciones reales en Arbitrum Sepolia y Arbitrum One.
- **Verificación del despliegue** — el bytecode desplegado, el owner y el cableado del registry se releen de cada red al margen del script de despliegue.

Modelo de amenazas

Se declara explícitamente porque cambia cómo deben leerse varios hallazgos. Quien dé por hecho que todo problema exige robar una clave juzgará mal los que no exigen absolutamente nada.

ACTOR	QUÉ SE ASUME
Dueño	De confianza. Tiene la passkey. El compromiso de la passkey queda fuera de alcance: equivale a ser el dueño de la wallet, y para ese caso existe la recuperación por guardianes.
Agente	**Semi-confiable, y a efectos de esta revisión se asume hostil.** Su clave vive en un servidor, en una variable de entorno o un fichero `.env`, corriendo desatendida; y el agente puede ser de un operador tercero que el dueño no controla. «El agente actúa fuera de su mandato» no es una condición previa exótica: es justamente el suceso que el sistema de permisos existe para acotar.
Guardián	Honesto en mayoría. La recuperación exige dos de tres; el diseño acepta que dos guardianes coaligados se queden la wallet si el dueño no cancela en 48 horas.
Tercero	No confiable, no tiene ninguna credencial, y puede llamar a todos los puntos de entrada abiertos —incluidas las factories— en cualquier orden y en cualquier red.
Protocolo	Se asume correcto en sí mismo, pero puede comportarse de forma adversarial en su interfaz: un pool puede estar controlado por el atacante, un token puede reentrar o devolver valores inesperados.

> **Por qué «hace falta la clave del agente» no es un atenuante.** Los topes por token, los presupuestos diario y vitalicio, la caducidad, la pausa y la lista de destinatarios existen precisamente porque el agente no es de fiar del todo. Si una clave de agente comprometida u hostil se tratara como fuera de alcance, toda la capa de permisos no tendría sentido. Los límites que solo se sostienen mientras el agente se porta bien no son límites: son sugerencias. Por eso BVCC-01 es Crítico: convierte «este agente puede gastar 0,003 ETH» en «este agente puede llevárselo todo», que es un fallo de la promesa central del producto y no de una función concreta. Las condiciones previas se recogen hallazgo a hallazgo en el §3 para que el lector pueda ponderar cada una.

Modelo de severidad

La severidad es **impacto × probabilidad**. El impacto es el peor desenlace para un titular si se explota el problema; la probabilidad pesa las condiciones previas que debe cumplir un atacante — compromiso de clave, una carrera, un timelock, una configuración concreta de la wallet— y no lo probable que se considere el ataque en abstracto.

SEVERIDAD	SIGNIFICADO
CRÍTICA	Pérdida de todos los fondos de la wallet, o toma de control completa, con condiciones previas que un atacante puede alcanzar de forma realista.

ALTA	Pérdida de fondos más allá de los límites configurados, o toma de control, pero condicionada a una carrera, un timelock, una acción del dueño o una configuración concreta.
MEDIA	Debilita un control de seguridad, degrada la recuperación o deniega el servicio, sin mover fondos directamente.
BAJA	Viola un invariante declarado con consecuencia práctica limitada.
INFORMATIVA	Sin impacto en el despliegue actual; endurecimiento, o una decisión de diseño que se registra para que no se deshaga en silencio.

La severidad de un hallazgo se indica tal como se encontró, no tal como queda tras la remediación. El estado se sigue por separado, de modo que un Crítico corregido sigue viéndose como un Crítico que se corrigió.

Limitaciones de esta revisión

Se declaran aquí y no al final, porque condicionan cómo debe leerse cada hallazgo. Esta es una revisión **interna**: quienes escribieron el código son quienes lo revisaron. No ha sido auditada externamente. Cuatro de los hallazgos salieron de una pasada posterior sobre el código de V3 con una herramienta de análisis distinta, y cada uno había sobrevivido a tres rondas anteriores: cambiar de herramienta encontró lo que repetir el método no encontraba. Aquella pasada también fue interna — **ningún tercero independiente ha revisado este código**. Que un hallazgo no aparezca en este documento es, por tanto, evidencia débil de que no exista. El §7 desarrolla qué establece y qué no establece esta revisión.

3 Registro de hallazgos

Trece hallazgos en el código actual, más dos cerrados en generaciones superadas (Anexo D). La severidad es la que tenían al encontrarlos; el estado es a fecha de 2026-07-29.

ID	HALLAZGO	REQUIERE	SEVERIDAD	ESTADO
`BVCC-01`	Reentrada cruzada vía `_currentAgent`	Clave del agente, u operador hostil	CRÍTICA	Corregido en V4
`BVCC-02`	La máscara de comandos del Universal Router acepta un depósito de puente como swap	Clave del agente, con acceso al router	CRÍTICA	Corregido antes de desplegar
`BVCC-03`	Los presupuestos de token no acotan el gasto en protocolos	Clave del agente + allowance viva	ALTA	ABIERTO
`BVCC-04`	Apropiación de guardianes a través de la factory abierta	**Ninguna** — cualquier tercero	ALTA	Corregido en V4
`BVCC-05`	Una recuperación completada no detenía a los agentes existentes	Un compromiso ya en curso	ALTA	Corregido en V4
`BVCC-06`	`credentialId` sin autenticar en el evento de la factory	**Ninguna** — cualquier tercero	MEDIA	Corregido en V4
`BVCC-07`	Credencial obsoleta tras una recuperación	**Ninguna** — tras cualquier recuperación	MEDIA	Corregido en V4
`BVCC-08`	Los guardianes no se podían rotar nunca	**Ninguna** — operativo	MEDIA	Corregido en V4
`BVCC-09`	Un solo guardián puede bloquear una recuperación indefinidamente	Un guardián malicioso	MEDIA	Aceptado
`BVCC-10`	Pausar o revocar un agente no retira sus allowances	Un spender ya aprobado	MEDIA	ABIERTO — interfaz
`BVCC-11`	`approve` podía llevar ETH saltándose la lista de destinatarios	Clave del agente	BAJA	Corregido en V4
`BVCC-12`	`increaseLiquidity` no comprueba el dueño en el NFPM de v3	Clave del agente; inalcanzable hoy	INFORMATIVA	Mitigado por exclusión
`BVCC-13`	Los constructores de los validators aceptan la dirección cero	**Ninguna** — solo error de despliegue	INFORMATIVA	Aceptado

Por severidad y por estado

SEVERIDAD	Nº
CRÍTICA	2
ALTA	3
MEDIA	5
BAJA	1
INFORMATIVA	2

ESTADO	Nº
Corregidos y desplegados	9
Abiertos	2
Aceptados y documentados	2

Cómo leer el registro. La severidad se recoge tal como estaba al encontrar el problema, así que un Crítico corregido sigue leyéndose como Crítico. Es deliberado: la pregunta del lector suele ser «de qué era capaz este código», no solo «qué queda». Nueve hallazgos están remediados en bytecode vivo en las seis redes, y los dos Críticos están entre ellos. El que debería condicionar el comportamiento de un dueño hoy es **BVCC-03**, que V4 no cierra. La columna **Requiere** importa tanto como la severidad: seis de los trece no necesitan credencial alguna —un desconocido que nunca tuvo una clave llega a BVCC-04 y a BVCC-06— mientras que cinco presuponen una clave de agente hostil o robada, que el modelo de amenazas del §2 trata como suceso esperable y no como excusa.

4 Hallazgos en detalle

Cada hallazgo se expone con la severidad que tenía al encontrarlo, las condiciones previas que necesita un atacante y la evidencia que lo estableció. La remediación y su verificación van en §5.

El código de V3 se sometió a otra pasada interna con una herramienta de análisis distinta, que planteó cuatro cuestiones. Las cuatro se reprodujeron contra el código en vez de darlas por buenas a partir de su descripción; los tests están en `test/ThirdPartyClaims.t.sol`. Las cuatro se sostienen, aunque dos son más estrechas de lo que se enunciaba y una es una propiedad de ERC-20 más que de esta wallet. Ninguna había aparecido en las tres rondas internas.

BVCC-01 **CRÍTICA** Corregido en V4

Reentrada cruzada vía `_currentAgent`

UBICACIÓN	`BVCCAgentWallet.sol` — `executeAsAgent` / `execute`
IMPACTO	Pérdida total de los fondos. Se saltan todos los límites configurados en una sola transacción; una prueba de concepto movió **49,925 ETH** contra un presupuesto vitalicio de 0,003 ETH.
PROBABILIDAD	Media. La dirección del agente tiene que adquirir código después de ser autorizada: una delegación EIP-7702 que el propio agente firma con su clave, o un despliegue CREATE2 a una dirección pre-comprometida. No hace falta ninguna acción del dueño ni hay aviso.
AFECTA A	V1, V2 y V3 por igual — confirmado drenando una wallet creada desde la factory V2 *viva* sobre un fork de Arbitrum One. Las wallets que no hayan migrado a V4 siguen expuestas.

Este resultado se descartó en un primer momento alegando que un agente es un EOA y que no se le puede ceder el control a mitad de transacción. Ese razonamiento no se sostiene, y el hallazgo queda confirmado.

Mecanismo

`executeAsAgent` fija `_currentAgent = agent` alrededor de su llamada a `super.execute()`, y `_erc7821AuthorizedExecutor` concede la vía `execute()` de nivel propietario a cualquier llamante que cumpla `caller == _currentAgent`. El `execute()` padre no lleva guard de reentrada — `nonReentrant` está solo en `executeAsAgent`— y no aplica ningún límite de agente, porque las comprobaciones viven en `executeAsAgent`. La premisa del EOA descansa en `AgentMustBeEOA` (`agent.code.length == 0`), que se exigía **solo en el momento de autorizar** y nunca se volvía a comprobar al ejecutar. Una dirección puede adquirir código después: mediante una delegación EIP-7702, firmada con la propia clave del agente —justo la clave que el modelo de amenaza asume que un atacante puede tener— o desplegando en una dirección CREATE2 precalculada que estaba sin código cuando se autorizó.

> **Alcance: afecta igual a V1, V2 y V3.** No es una regresión de V3. Los cuatro ingredientes —la bandera, el override del executor, el control de EOA solo al autorizar y el `execute()` padre sin guard— están literalmente en las tres generaciones; lo único que se movió son los números de línea. Se confirmó contra código desplegado, no contra el fuente: sobre un fork de Arbitrum One, una wallet creada desde la **factory V2 viva** (`0x8D9e2402…`) fue drenada con el mismo ataque (`test/LegacyV2ReentrancyFork.t.sol`).

Reproducción

`test/AgentReentrancyPoC.t.sol`. Se autoriza un agente sin código con topes de 0,001 / 0,002 / 0,003 ETH (por tx / diario / vitalicio). Después se coloca código en la dirección del agente, modelando la delegación. El agente manda un batch cuyo único elemento es una transferencia de **1 wei** a sí mismo —dentro de todos los topes— que le cede el control; desde ahí llama a `execute()` de vuelta y saca **49,925 ETH** de un saldo de 100 ETH. La diferencia de 0,075 ETH es la comisión propia del 0,15% de la wallet, lo que confirma que el drenaje viajó por la vía de propietario.

Radio de impacto

Se saltan de golpe todas las restricciones del agente: por transacción, diaria, de período y vitalicia, la lista blanca de destinatarios y las call policies de V3. Convierte un agente acotado en control equivalente al del propietario, un agujero estrictamente mayor que el vector de exfiltración que V3 vino a cerrar. No afecta a la vía biométrica del propietario, ni permite nada a un tercero que no tenga la clave del agente.

Remediación — dos capas

Se implementaron tres candidatos y se midieron contra tres formas de ataque, en vez de discutirlos. S1 manda 1 wei directamente al agente; S2 deja al agente sin código al entrar y hace que el propio batch le despliegue código a esa dirección mediante un ayudante whitelisteado (sin EIP-7702 de por medio); S3 no nombra al agente en ningún momento y le hace llegar valor a través de un intermediario permitido.

CANDIDATO	S1 DIRECTO	S2 CREATE2	S3 INDIRECTO	TAMAÑO FACTORY
Ninguno (antes del fix)	drena	drena	drena	`24.283`
B — re-chequear `code.length`	bloquea	**drena**	bloquea	`24.294`
C — el batch no puede apuntar al agente	bloquea	bloquea	**drena**	`24.312`
A — guard en el `execute()` externo	bloquea	bloquea	bloquea	`24.317`
A + B (lo que se publica)	bloquea	bloquea	bloquea	`24.328`

Los dos candidatos descartados fallan por motivos instructivos. **B por sí sola** es otra comprobación puntual —la misma clase de suposición que causó el bug— y S2 se le cuela haciendo que el código aparezca *durante* el batch. **C por sí sola** cierra únicamente la puerta que usó la primera prueba de concepto; S3 llega al agente a través de un protocolo legítimo.

Lo que se publica son las dos capas. La **capa 1** vuelve a comprobar el invariante del EOA en cada ejecución, de modo que se exige de forma continua y no una sola vez. La **capa 2** sobreescribe `execute()` en la wallet de agente para rechazar cualquier entrada externa mientras hay un batch de agente en vuelo (`ReentrantAgentExecute`); `executeAsAgent` llega al padre por `super.execute()`, que resuelve estáticamente y por tanto salta el override, así que el camino legítimo queda intacto. La capa 2 es la que no depende de *cómo* ni de *cuándo* consiguió código el agente: cierra la ventana en sí. Cada capa tiene su propio test de regresión, y la capa 2 queda probada de forma independiente con la forma S2, donde la capa 1 pasa limpiamente y el drenaje se detiene igual.

Coste: la factory de agentes crece 45 bytes hasta 24.328, dejando 248 de margen bajo EIP-170, y `executeAsAgent` en caso 3 sube de 64.758 a 64.903 gas —145 gas, un 0,2%, porque la dirección del agente ya está caliente al ser quien firma la transacción—. Una consecuencia deliberada de comportamiento: un agente que adopte una delegación EIP-7702 por motivos ajenos dejará de funcionar hasta que la quite. Es el invariante que el propio contrato declara, ahora exigido.

> **Desplegado en V4 — pero solo para las wallets creadas en V4.** El arreglo cambia el bytecode de la wallet y por tanto las direcciones CREATE2, así que no puede alcanzar a una wallet ya desplegada; que `Create2Consistency.t.sol` fallara contra las constantes viejas fue la prueba mecánica de eso. Las factories V4 están vivas en las seis redes, **y toda wallet que siga en V1, V2 o V3 ejecuta bytecode vulnerable hasta que su dueño migre. Mitigación para esas wallets, sin redesplegar:** autorizar a los agentes con una `allowedRecipients` no vacía que no contenga la dirección del propio agente, lo que le quita al batch la capacidad de devolverle el control (`test_RecipientWhitelist_BlocksTheEntry`). Las wallets que se queden en el default permisivo —lista de destinatarios vacía, es decir, cualquier destino— son la configuración expuesta. Pausar o revocar al agente también lo cierra de inmediato.

Entre los Low, 20 `calls-loop` y 4 `timestamp` son inherentes a la ejecución por batches y a los límites indexados por día; el `return-bomb` de `_tryBalanceOf` está acotado por el mismo `PROBE_GAS_CAP` que introdujo V2, que también limita cuántos datos de retorno puede permitirse producir un token hostil. Un resultado sí es una carencia real, aunque inerte:

> **Detalle aceptado — sin zero-check en los constructores de los validators.** Ni `BVCCUniversalRouterValidator` ni `BVCCPositionManagerValidator` rechazan una dirección cero para el protocolo al que se atan, a diferencia de los constructores de las factories endurecidos en V1 (`ZeroOwner`). Un validator atado a la dirección cero denegaría todas las llamadas —fail-closed, así que el comportamiento desplegado es seguro— y las seis instancias desplegadas llevan los argumentos de constructor correctos, registrados con sus codehashes en `contracts/deployments/` y verificados en los exploradores. Añadir el control cambiaría el bytecode, y por tanto las direcciones CREATE2, forzando un redespliegue y un ciclo de gobernanza de 48 horas por red. Queda anotado aquí para la próxima revisión de bytecode, no ejecutado ahora.

> **Re-ejecución tras los arreglos de V4: 53 resultados — 3 High, 27 Low, 23 Informational.** Los tres High no cambian, y los tres son ya falsos positivos: los dos `uninitialized-state` son mappings, y `reentrancy-eth` sigue viendo la bandera escrita tras la llamada externa aunque el BVCC-01 cerrara la ventana — ahora incluso lista `execute()` entre las funciones alcanzables, que es precisamente donde está el guard. Apareció un segundo `dead-code` por `_afterRecovery()`, el hook de recuperación del BVCC-05: el mismo falso positivo que `_feeNumerator()`, una función virtual vacía que sobreescribe el hijo. Ninguna de las dos se toca. Los arreglos no introdujeron nada nuevo. **El análisis estático detecta patrones, no emite un veredicto**: esta cifra no se movió mientras la reentrada fue explotable durante tres generaciones, y tampoco se movió cuando el arreglo la cerró.

Los resultados Informational son de las mismas clases que en las pasadas de V1 y V2: ensamblador inline, low-level calls, convenciones de nombres en los inmutables, dispersión de pragmas por el árbol de OpenZeppelin, y el aviso de "dead code" sobre `_feeNumerator()` — falso positivo conocido: se alcanza por dispatch virtual y fija la comisión de agente del 0,15%.

BVCC-02 **CRÍTICA** Corregido antes de desplegar

La máscara de comandos del Universal Router acepta un depósito de puente como swap

UBICACIÓN	`BVCCUniversalRouterValidator.sol` — decodificación de comandos
IMPACTO	Crítico. El router habría enviado los fondos de la wallet a un atacante en otra cadena mientras el validator clasificaba la llamada como un swap normal: justo la clase de exfiltración que el validator existe para impedir.
PROBABILIDAD	Media — una clave de agente comprometida con acceso al Universal Router, y nada más.
NOTA	Encontrado en revisión **antes** de registrar el validator en ninguna red, así que nunca fue explotable en producción. Se registra porque lo interesante es el casi-fallo: el validator se escribió para impedir justo esto y lo habría permitido.

> **Bug crítico detectado en revisión — la máscara de comandos.** La primera versión del validator enmascaraba los bytes de comando con `& 0x3f`. El router real enmascara con `& 0x7f` (el bit `0x80` es `FLAG_ALLOW_REVERT`), lo que significa que `0x40` **no** es un alias de `0x00`: es `ACROSS_V4_DEPOSIT_V3`, un depósito en un puente cross-chain. Con la máscara equivocada el validator lo habría aceptado como un swap v3 mientras el router bridgeaba los fondos de la wallet hacia un atacante en otra cadena: exactamente la clase de exfiltración que V3 existe para impedir. La corrección sustituyó la máscara por coincidencia exacta, ató el validator a un único router y rechazó las cadenas de comandos vacías. El comportamiento se fuzzea ahora sobre los 256 bytes de comando contra un modelo del router, y se volvió a confirmar en mainnet tras el despliegue (ver Anexo B).

BVCC-03 **ALTA** **ABIERTO**

Los presupuestos de token no acotan el gasto en protocolos

UBICACIÓN	`BVCCAgentWallet.sol` — `_validateExecutionItem`, caso 3
IMPACTO	Crítico donde aplica: pérdida de la allowance viva entera, con independencia del presupuesto de token configurado para el agente.
PROBABILIDAD	Media-baja. Exige a la vez una clave de agente comprometida *y* una allowance que el dueño ya haya concedido a un protocolo alcanzable por el agente. Una wallet sin allowance viva no está expuesta.
NOTA	El único hallazgo de este informe que V4 no cierra. Ver §6 para la mitigación y el plan escalonado.

Los contadores por token solo acumulan para los selectores `transfer` y `approve` de ERC-20: a `_checkTokenWhitelistAndAmount` se llega desde esas dos ramas y desde ninguna más. Una llamada de caso 3 gasta por tanto tokens sin tocar `tokenMaxAmounts`, el límite diario ni el presupuesto vitalicio — y sin comparar el activo que viaja dentro de su calldata con `allowedTokens`. Esto es deliberado hasta cierto punto: el modelo cobra el presupuesto en el paso del `approve`, que es lo que afirma la batería de Aave sobre fork, así que en el flujo habitual de aprobar y luego operar el presupuesto sí se consume.

El hueco son las **allowances previas**, y el caso común no es rebuscado: un propietario que haya concedido una allowance amplia a un router o a Permit2 desde la interfaz —práctica estándar— deja al agente moviendo ese token por la vía de caso 3 indefinidamente sin consumir una unidad de su presupuesto. Reproducido en `test_Claim2_PreExistingAllowanceEscapesTokenBudgets`: con los tres topes de token puestos a

10, el agente mueve 500 a través de un router whitelisteado y ambos contadores se quedan a cero. El propio `approve(address,address,uint160,uint48)` de Permit2 es asimismo un selector distinto y tampoco cuenta.

> **Corrección de severidad.** Un borrador anterior de esta sección describía la pérdida como operar por encima del volumen previsto, apoyándose en que las call policies anclan todos los destinatarios a la wallet. Eso se queda corto. El anclaje garantiza *dónde aterriza el output*, no *cuánto vale*: un agente comprometido puede cambiar un activo valioso por uno sin valor a través de un pool que él controla, y el token inútil vuelve obedientemente a casa mientras el valor se queda en la liquidez del atacante. Nada en el diseño actual se opone — el validator del Universal Router no contiene ni una sola referencia a cantidades o precios, y las policies de SwapRouter02 anclan el destinatario y nada más, así que un swap con `amountOutMinimum = 1` pasa todos los controles. Donde exista una allowance viva, esto es un vaciado completo de los tokens que cubra, no deslizamiento de precio.

La selección de tokens lo estrecha, pero no lo cierra. Elegir qué tokens puede tocar un agente sí gobierna `transfer` y `approve`, de modo que un token no seleccionado y sin allowance viva no se puede mover: el agente tampoco puede aprobarlo. Quedan dos huecos. Un token que ya arrastra una allowance es alcanzable al margen de la selección. Y el `borrow` y el `withdraw` de Aave no consumen allowance alguna, así que ahí no hay nada que poner a cero: un agente con policy de Aave puede endeudar la wallet en un activo que el propietario nunca seleccionó, contra su propio colateral, con el riesgo de liquidación que eso implica. Los fondos aterrizan en la wallet —es daño, no robo— pero es justo el daño que la selección de tokens debería haber evitado.

Plan de remediación

La contabilidad completa dentro de la wallet exigiría que los validators profundos decodifiquen y devuelvan los pares `(token, gasto máximo)` que ven, acumulándolos la wallet por `_applyTokenSpend` y contando `amountInMaximum` en las rutas de salida exacta. Eso toca todos los validators más la wallet, y la factory de agentes tiene 101 bytes de margen bajo EIP-170. Por eso se escalona en vez de intentarse ahora.

FASE	MEDIDA	COSTE
Ahora solo interfaz	Renombrar el límite para que deje de sugerir un tope global — gobierna transferencias y autorizaciones directas, no el volumen DeFi	Ninguno
	Al autorizar, leer las allowances de cada token relevante hacia cada protocolo que las policies del agente puedan alcanzar, y exigir una revocación firmada de las que no sean cero	Frontend + un batch firmado
	Ejecutar las operaciones del agente como batches atómicos: `approve(exacto)` → llamada al protocolo → `approve(0)`, de modo que la primera pata consuma presupuesto y nada sobreviva al batch	Frontend; funciona con el contrato actual
	Para Permit2, comprobar las dos capas —la allowance ERC-20 hacia Permit2 y la allowance interna de Permit2 hacia el router— con cantidades exactas y expiraciones cortas	Frontend
	Una pantalla de emergencia para el dueño que revoque todas las allowances, y saldos moderados en las wallets que lleven agentes	Frontend + operativa

Después capa de validators	Control de precio en los validators de swap: rechazar una ruta cuyo `amountOutMinimum` sea inverosímil frente a un oráculo o un TWAP. Un validator es una función `view`, así que puede consultarlo. **Esta es la medida que cierra el vaciado por pool amañado.**	Validators nuevos; 48 h de gobernanza por red
	Un validator de Aave que compruebe el argumento `asset` contra los tokens seleccionados de la wallet, para que la selección gobierne también el préstamo	Validator nuevo; mismo ciclo
Más adelante	Un vault de agente por el que el agente esté obligado a pasar, de modo que la cantidad entregada sea un techo físico y los routers salgan de `allowedProtocols`	Contrato nuevo + integración
	Contabilidad dentro de la wallet, cuando un rediseño de la factory libere el bytecode necesario	Trabajo de escala V5

Cómo baja la severidad. La calificación viene de dos cosas: la pérdida no tiene techo, y la condición previa es comportamiento normal del usuario. Fíjese en lo que ocurre cuando no existe ninguna allowance viva — el agente tiene que hacer `approve` antes de poder gastar, y ese approve *sí* se cuenta contra el presupuesto por token y pasa por las listas de tokens y destinatarios. El presupuesto vuelve a acotarlo todo. El hallazgo no es por tanto «el presupuesto no funciona» sino «el presupuesto no funciona *mientras exista una allowance viva*», y esa condición es atacable sin tocar los contratos.

ESTADO	QUÉ QUEDA ACOTADO	SEVERIDAD
Hoy	Nada, si existe una allowance	**ALTA**
+ higiene de allowances (a cero al autorizar, approvals exactos y atómicos, expiraciones cortas en Permit2)	El gasto acumulado, vía el presupuesto de token	MEDIA
+ control de precio en los validators de swap	También la pérdida por operación	BAJA

La higiene sola se queda en media porque es una foto: el dueño puede conceder una allowance desde cualquier dApp después. Baja la probabilidad; solo el validator baja el impacto. Dos salvedades: el `borrow` y el `withdraw` de Aave no consumen allowance, así que la higiene no llega ahí hasta que un validator de Aave compruebe el `asset`; y un límite de precio acota cada operación, no el total — acotar el gasto acumulado de caso 3 exige la contabilidad en la wallet que el bytecode no puede albergar hoy.

Lo que las medidas de interfaz no pueden hacer. Impedir approvals ilimitados mientras haya un agente activo no es aplicable desde el frontend de la propia wallet: el propietario puede conceder una allowance desde cualquier dApp, y nada on-chain lo impide. La comprobación en el momento de autorizar es por tanto una foto que cierra el caso común —la allowance olvidada— no un invariante. Solo la capa de validators lo convierte en uno.

BVCC-04 ALTA Corregido en V4

Apropiación de guardianes a través de la factory abierta

UBICACIÓN `BVCCWalletFactory.sol` / `BVCCAgentWalletFactory.sol` — `createWallet`; `BVCCWallet.sol` — `setGuardians`

IMPACTO	Crítico. Toma de control completa: los guardianes del okupa rotan al dueño en cuanto pasa el timelock de recuperación.
PROBABILIDAD	Baja. El atacante tiene que desplegar la dirección de la víctima primero en una red que esta no haya usado, la víctima tiene que fondearla, y deben pasar las 48 horas sin que el dueño cancele. Detectable leyendo `guardians()` antes de meter fondos.
REPORTADO POR	Pasada interna posterior, herramienta distinta

La dirección CREATE2 de una wallet se deriva únicamente de la clave pública de la passkey; los guardianes no forman parte de ella. `createWallet` no exige autorización alguna y fija los guardianes en la misma llamada, y `setGuardians` es de un solo uso. Quien despliegue una dirección primero elige, de forma permanente, a las tres partes que pueden rotar a su propietario. Si la wallet ya existe, la factory la devuelve sin cambios, así que quien llegue segundo recibe la wallet ocupada sin error y sin ninguna señal.

La apropiación completa se reprodujo de principio a fin (`test_Claim1_GuardianSquattingTakesOverTheWallet`): un atacante despliega la dirección determinista de la víctima con sus propios guardianes; la app de la víctima «crea» después la wallet y recibe de vuelta esa misma dirección; dos de los guardianes del atacante abren una recuperación, agotan el timelock de 48 horas y rotan el signer. La wallet y su saldo son suyos.

El diseño multichain amplifica esto precisamente por ser determinista: la clave pública de la passkey queda visible on-chain en cuanto el usuario despliega en cualquier red, y la misma dirección es reclamable en todas las demás. La exposición son, por tanto, **las redes donde un usuario aún no ha desplegado**, no las wallets ya en uso — una vez fijados los guardianes correctos no se pueden sustituir. El timelock de 48 horas junto con `cancelRecovery` es una defensa real, pero solo en una red que el propietario esté vigilando, lo que por definición excluye aquellas a las que no ha desplegado.

> **Una parte de la recomendación del revisor no se sostiene.** Rechazar guardianes duplicados on-chain es higiene, no un agujero: las aprobaciones se llevan por dirección, así que un guardián duplicado no puede aprobar dos veces ni alcanzar por sí solo el umbral de 2 de 3 (`test_Claim1_DuplicateGuardianCannotSelfApprove`). Los duplicados degradan el esquema a 2 de 2; no permiten una apropiación en solitario.

Corrección, ya aplicada en código. La factory ya no elige guardianes: despliega la wallet sin ninguno, y `setGuardians` queda protegida con `msg.sender == address(this)` —el mismo patrón de auto-llamada que ya usan `authorizeAgent` y `setCallPolicy`— de modo que solo una operación firmada con la passkey puede fijarlos. Los duplicados también se rechazan ahora. El ocupante se queda con un cascarón vacío que no puede configurar y, de hecho, ha pagado el gas del despliegue de la víctima. Los guardianes no se leen en ningún sitio salvo en la ruta de recuperación, así que una wallet sin ellos opera con normalidad; lo único no disponible es la recuperación hasta que el dueño firme. Lo cubren siete tests de regresión, incluidos la vía real del propietario —un batch de `execute()` cuyo item apunta a la propia wallet— y el hecho de que un agente autorizado tampoco llega, porque `executeAsAgent` rechaza cualquier item dirigido a la wallet.

BVCC-05 ALTA Corregido en V4

Una recuperación completada no detenía a los agentes existentes

UBICACIÓN	`BVCCWallet.sol` — `executeRecovery`; `BVCCAgentWallet.sol` — `_afterRecovery`

IMPACTO	Alto. El agente del atacante sigue gastando después de que el dueño crea haberlo dejado fuera — y el compromiso es justo el motivo por el que un dueño recurre a la recuperación.
PROBABILIDAD	Media — aplica a cualquier recuperación hecha por un compromiso, que es la razón principal de que el mecanismo exista.

La recuperación no revocaba los agentes. CORREGIDO EN V4 `executeRecovery` rotaba el signer y no tocaba nada más, así que un agente autorizado antes de la recuperación seguía gastando después, con su presupuesto intacto. Importa porque el compromiso es justamente el motivo por el que un dueño recurre a la recuperación: quien recuperaba creyendo haber echado al atacante no le había revocado el agente. Ahora la recuperación pausa los agentes mediante un hook `_afterRecovery`. Pausar en vez de revocar es deliberado: gas constante sea cual sea la lista de agentes, reversible, y conserva la configuración de cada uno para que el nuevo dueño la juzgue en vez de tener que reconstruirla. La guarda es `if (!paused()) _pause()`: sin ella, una wallet con los agentes ya pausados no habría podido completar una recuperación.

BVCC-06 MEDIA Corregido en V4

`credentialId` sin autenticar en el evento de la factory

UBICACIÓN	Evento `WalletCreated` de la factory; `BVCCWallet.sol` — `_setCredential` / `setCredentialId`
IMPACTO	Medio. Un okupa podía publicar una credencial falsa para la dirección de otro, y tras una recuperación la credencial on-chain seguía apuntando a la passkey recién sustituida: la interfaz ofrecería la clave equivocada.
PROBABILIDAD	Media — el primero se deriva de BVCC-04; el segundo, de cualquier recuperación completada.
REPORTADO POR	Pasada interna posterior, herramienta distinta — revisión de seguimiento

Al corregirlo aparecieron dos hallazgos emparentados. El `credentialId` viajaba por la factory como campo de evento sin autenticar, así que un ocupante podía publicar uno falso para la dirección de otro. No puede costar fondos —las firmas se validan contra la clave pública guardada— pero un cliente que se fíe preselecciona una credencial que el dueño no tiene, y el prompt de la passkey falla. Se ha sacado de la factory por completo: ahora lo anuncia la propia wallet, en la misma llamada firmada que fija los guardianes, mediante `CredentialSet(bytes32 indexed credentialHash, bytes credentialId)`. El hash va indexado para que un cliente con el `rawId` de una passkey pueda encontrar la wallet a la que pertenece.

BVCC-07 MEDIA Corregido en V4

Credencial obsoleta tras una recuperación

UBICACIÓN	`BVCCWallet.sol` — `executeRecovery` / `setCredentialId`
IMPACTO	Medio. Tras una recuperación por guardianes la credencial on-chain seguía apuntando a la passkey que el dueño ya no tenía, así que un cliente que se fíe pide la clave equivocada y la wallet recuperada parece inservible.
PROBABILIDAD	Segura tras cualquier recuperación completada: no es un ataque, es un estado final roto del mecanismo que existe para rescatar la wallet.
REPORTADO POR	Pasada interna posterior, herramienta distinta — revisión de seguimiento

Detectado al corregir BVCC-04. `executeRecovery` rota el signer pero dejaba la credencial intacta, de modo que tras una recuperación por guardianes la credencial on-chain apuntaba a una passkey que el dueño ya no tenía; `setCredentialId`, también con auto-llamada, lo cierra.

BVCC-08 MEDIA Corregido en V4

Los guardianes no se podían rotar nunca

UBICACIÓN	`BVCCWallet.sol` — `setGuardians`
IMPACTO	Medio. Un guardián cuya clave se perdiera o se comprometiera después era permanente durante toda la vida de la wallet; el único remedio era crear otra wallet y mover los fondos.
PROBABILIDAD	Baja como ataque, segura como fallo operativo en un horizonte suficientemente largo.

Los guardianes no se podían rotar nunca. CORREGIDO EN V4 `setGuardians` era de un solo uso, así que un guardián cuya clave se perdiera o se comprometiera después era permanente durante toda la vida de la wallet, y el único remedio era crear otra wallet y mover los fondos. El conjunto es ahora reemplazable por el dueño. Sigue siendo una auto-llamada, de modo que la apropiación del BVCC-04 continúa cerrada, y la rotación se rechaza con una recuperación en vuelo (`RecoveryActive`) — si no, una passkey robada podría cambiar los guardianes por debajo de una recuperación ya en marcha. El dueño cancela primero y rota después.

BVCC-09 MEDIA Aceptado

Un solo guardián puede bloquear una recuperación indefinidamente

UBICACIÓN	`BVCCWallet.sol` — `initiateRecovery`
IMPACTO	Medio. Denegación de servicio contra el mecanismo de rescate. No se mueve ningún fondo y el dueño no se rota nunca.
PROBABILIDAD	Media — exige un guardián malicioso o comprometido dispuesto a hacer front-run a cada segunda aprobación.
NOTA	Aceptado como limitación documentada: es una carrera, no un bloqueo, y en cuanto el par honesto consigue dos aprobaciones nadie puede reiniciarlo.

Un solo guardián puede bloquear una recuperación indefinidamente.
ABIERTO — ACEPTADO `initiateRecovery` se puede llamar siempre que haya menos de dos aprobaciones, y sobrescribe la clave pendiente y pone el timelock a cero. Un guardián malicioso solo tiene por tanto que adelantarse a cada segunda aprobación para que la pareja honesta no alcance nunca el umbral. Es denegación de servicio contra el mecanismo de rescate, no robo: el dueño no llega a rotar. Y es una carrera, no un candado — un test acompañante demuestra que en cuanto la pareja honesta mete dos aprobaciones, ya nadie puede reiniciarla. Aceptado como limitación documentada.

> **El modelo de confianza, sin adornos.** Dos guardianes coludidos se quedan con la wallet si el dueño no cancela dentro de la ventana de 48 horas — eso es lo que hace que la recuperación funcione, y así está probado. Implica que el dueño debe vigilar todas las redes donde exista su wallet, que es el mismo requisito operativo que la apropiación de guardianes del BVCC-04 dejó al descubierto por otra vía.

BVCC-10 MEDIA ABIERTO — interfaz

Pausar o revocar un agente no retira sus allowances

UBICACIÓN	Semántica de allowance de ERC-20; flujo de revocación en la interfaz
IMPACTO	Medio. Un dueño que pausa un agente comprometido no ha detenido al spender que ya aprobó. La allowance sobrevive en el contrato del token.
PROBABILIDAD	Media — toda revocación hecha bajo la suposición razonable de que basta con eso.
NOTA	Inherente a ERC-20: nada que haga la wallet puede retirar una allowance que guarda el token. El remedio es una interfaz que revoque explícitamente — ver §6.

Una allowance vive en el contrato del token, así que nada de lo que haga la wallet puede retirarla. Reproducido en `test_Claim3_AllowanceSurvivesPauseAndRevoke`: el agente aprueba dentro de su tope, el propietario pausa *y* revoca, y el spender se lleva igualmente el importe completo. El revisor señala correctamente que el importe se cargó al presupuesto en el momento de aprobar, así que no se elude ningún límite. Es una propiedad de cualquier smart account, no un defecto de esta, y afecta a la respuesta ante incidentes más que al modelo de permisos: tras revocar un agente, el propietario debería además poner a cero las allowances que le concedió. Un aviso explícito al pausar y una pantalla de revocación de allowances lo cubren.

BVCC-11 **BAJA** **Corregido en V4**

`approve` podía llevar ETH saltándose la lista de destinatarios

UBICACIÓN	`BVCCAgentWallet.sol` — `_validateExecutionItem`
IMPACTO	Bajo. Viola el invariante que el propio contrato declara —las llamadas de token no llevan valor—; la consecuencia práctica está acotada por el presupuesto de ETH, que sigue aplicando.
PROBABILIDAD	Baja — exige una clave de agente comprometida y un token que acepte llamadas con valor.
REPORTADO POR	Pasada interna posterior, herramienta distinta

La severidad queda limitada por un detalle que el revisor no menciona: el `approve` del destino tiene que ser `payable` para que la llamada llegue a funcionar. Los ERC-20 estándar no lo son, así que el intento simplemente revierte; el test necesita un token payable fabricado a propósito para demostrar la vía. Explotarlo exige por tanto que el propietario haya whitelisteado un token inusual u hostil. Al lado hay una desprolijidad emparentada: un `transfer` que lleve `value` lo ejecuta el padre sin reenviar el ETH, y aun así el agente carga ese valor a su presupuesto de ETH. Ambas quedan ahora rechazadas: una llamada de token que lleve valor revierte con `TokenCallWithValue`, verificado por un test de regresión que además confirma que un `approve` normal, sin valor, sigue funcionando.

BVCC-12 **INFORMATIVA** **Mitigado por exclusión**

`increaseLiquidity` no comprueba el dueño en el position manager v3

UBICACIÓN	`NonfungiblePositionManager` de Uniswap v3; presets de call policy
IMPACTO	Alto si estuviera en la whitelist: a diferencia de `decreaseLiquidity`, `collect` y `burn`, el position manager no comprueba el dueño en `increaseLiquidity`. Un agente podría llevar los tokens aprobados de la wallet a una posición del atacante — un bypass de la lista de destinatarios que no se puede anclar, porque la palabra 0 es el `tokenId`, no una dirección.

PROBABILIDAD	No alcanzable tal como está desplegado: el selector está deliberadamente fuera de todos los presets de policy.
NOTA	Se registra para que nadie deshaga la exclusión añadiendo el selector por comodidad. Para añadir liquidez, el agente mintea una posición nueva con el destinatario anclado a la wallet. Reactivarlo exige un validator profundo que compruebe `ownerOf(tokenId) == wallet`.

BVCC-13 **INFORMATIVA** **Aceptado**

Los constructores de los validators aceptan la dirección cero

UBICACIÓN	`BVCCUniversalRouterValidator.sol`, `BVCCPositionManagerValidator.sol` — constructores
IMPACTO	Ninguno tal como está desplegado. Un validator construido con router o position manager a cero no casaría con nada y denegaría todas las llamadas: un fallo en la dirección segura.
PROBABILIDAD	Solo error en el despliegue; no lo controla un atacante.
NOTA	Se releyó cada instancia desplegada en las seis redes y ambos inmutables son distintos de cero. Aceptado en vez de corregido: añadir la comprobación obligaría a redesplegar y volver a registrar validators que son demostrablemente correctos.

5 Remediación y verificación del despliegue

Nueve hallazgos están remediados. Siete exigían cambiar el bytecode y viajaron juntos como la generación V4; los otros dos se cerraron antes de que el código afectado llegara a ninguna red. Esta sección recoge qué está vivo, dónde, y cómo se comprobó al margen del script de despliegue.

Siete hallazgos exigían cambiar el bytecode. Los siete están corregidos y **vivos en las seis redes**. Como las direcciones CREATE2 se derivan del bytecode, no se podían aplicar a las wallets ya desplegadas: la migración es el mismo playbook que V1→V2 y V2→V3 —los usuarios recrean su wallet y mueven fondos, tras lo cual se hace `kill()` de las factories sustituidas—. Viajaron en un solo despliegue en vez de forzar varias migraciones. Los contratos se renombraron a V4 y el dominio EIP-712 va detrás (`BVCCSmartWalletV4`), igual que en la generación V2→V3; la interfaz lee ese dominio para detectar una wallet obsoleta y ofrecer la migración.

ELEMENTO	ESTADO
Reentrada cruzada (BVCC-01)	**Corregido** dos capas, cada una con su test de regresión
Apropiación de guardianes (BVCC-04)	**Corregido** la factory ya no los elige; `setGuardians` exige passkey
`credentialId` sin autenticar (BVCC-06)	**Corregido** fuera de la factory, a un evento firmado que emite la wallet
Credencial obsoleta tras recuperación (BVCC-07)	**Corregido** `setCredentialId`, con auto-llamada
`approve` con ETH (BVCC-11)	**Corregido** las llamadas de token deben llevar valor cero
Los agentes sobreviven a la recuperación (BVCC-05)	**Corregido** la recuperación pausa los agentes
Los guardianes no se podían rotar (BVCC-08)	**Corregido** el dueño puede reemplazarlos, nunca durante una recuperación
Contabilidad de token en caso 3 (BVCC-03)	**Abierto** escalonado; la fase de validators es la que lo cierra
Bloqueo de recuperación por un guardián (BVCC-09)	**Abierto** aceptado; denegación de servicio, no robo
Zero-check en constructores de validators (BVCC-13)	**Abierto** aceptado; inerte — se releyó cada instancia desplegada y ambos inmutables son distintos de cero
UX de revocar allowances y defaults de topes (BVCC-10)	**Abierto** trabajo de interfaz — deliberadamente fuera del contrato

Despliegue de V4 — Arbitrum One · Ethereum · BNB Chain · Base · Polygon · Arbitrum Sepolia

CONTRATO	DIRECCIÓN (CREATE2, IDÉNTICA EN TODAS LAS REDES)
SmartWalletFactory V4 (salt …0A)	`0xfd105197109244483b5f870501326E6faec9F93c`
AgentWalletFactory V4 (salt …0B)	`0xf3A61F9d64d45362E149A111289546523BCd26a6`
ValidatorRegistry	`0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2 — sin cambios`

Verificado al margen del script de despliegue, releyendo cada red: el bytecode desplegado es idéntico en las seis (smart wallet 15.959 bytes, agent wallet 24.541), el owner es la wallet hardware de gobernanza `0x3145Bd5e…A4c` y el registry está intacto. Las factories están verificadas en los exploradores.

El registry queda deliberadamente intacto: es la dirección que va compilada como constante dentro de cada wallet, y el test de congelación demuestra que la wallet V4 sigue resolviendo a ella. Los validators se comparten entre generaciones, así que no hubo que volver a registrar nada para V4.

Liquidez en v4 — ciclo completo ejecutado por un agente en mainnet

La ruta de validación profunda que introdujo V3 y hereda V4 se recorrió entera con un agente en Arbitrum One, contra un pool nativo: el caso que el anclaje por bitmap no sabe expresar, y la razón de ser de `BVCCPositionManagerValidator`. Pool ETH/USDC, fee 500, tick spacing 10, sin hooks, rango completo.

PASO	TRANSACCIÓN	GAS	RESULTADO
mint	`0xb4245927…f1269`	622.967	tokenId 190877, `ownerOf` = wallet
lectura de la posición	—	—	currency0 nativo, currency1 USDC, liquidez 4.377.897.532
collect	`0x9296f6f3…a865c7`	218.749	correcto, cero proceeds, cero fee
decrease 100% + burn	`0x92cce43d…dca9ef`	267.739	NFT quemado

El validator se comportó como estaba diseñado en todo el ciclo: las acciones `TAKE_PAIR` y `SWEEP` se aceptaron con `recipient` resolviendo a la wallet, y el dry-run pasó en cuanto el PositionManager entró en `allowedProtocols`. El mint envió 0,0001 ETH y el `SWEEP` devolvió los 17.050 wei de sobrante nativo, de modo que el PositionManager se quedó con **cero ETH**: justo el caso de valor varado que el validator rechaza cuando ningún `SETTLE` nativo consume el value. Estado final releído en cadena para este informe: `ownerOf(190877)` revierte `NOT_MINTED`, la liquidez de la posición es 0 y el saldo nativo del PositionManager es 0.

> Este ciclo se ejecutó **solo en Arbitrum One**. En las otras cinco redes los validators están desplegados, registrados y confirmados como atados al PositionManager correcto, pero el ciclo de liquidez de extremo a extremo no se ha ejecutado. Lo que afirma este informe es, por tanto: probado de extremo a extremo en Arbitrum, desplegado y verificado idéntico en el resto.

La factory de agentes queda en **24.541 bytes**, 35 por debajo del límite EIP-170, con `executeAsAgent` en caso 3 a 64.972 gas. Ese margen es estrecho, y es la razón de que dos medidas de endurecimiento candidatas se dejaran deliberadamente fuera del contrato: exigir topes de token no nulos cuando un agente tiene acceso a protocolos, y limitar las aprobaciones del propio dueño mientras haya un agente activo. Las dos pertenecen a la interfaz, donde no cuestan nada y pueden explicarse. El bytecode restante queda reservado para medidas sin equivalente fuera de la cadena.

6 Riesgo residual y recomendaciones

Cuatro hallazgos no los cierra V4. Uno es el riesgo material para un titular hoy; los otros tres son limitaciones aceptadas o trabajo de interfaz. Esta sección dice qué debería hacer un dueño en la práctica.

El que importa: BVCC-03

V3 ancló el destino de las llamadas a protocolo del agente, y eso se sostiene. Pero anclar un destino no es lo mismo que acotar un valor. Donde el dueño ya ha concedido una allowance de token a un protocolo que el agente alcanza, una clave de agente comprometida puede cambiar un activo valioso por uno inútil en un pool que el atacante controla: el ancla de destino manda el resultado a casa y el valor se queda con el atacante. Ningún validator mira cantidades ni precios.

Qué debería hacer un dueño hoy, sin tocar contratos: poner a cero cualquier allowance viva hacia los protocolos que el agente pueda alcanzar *antes* de autorizarlo, y preferir aprobaciones por transacción a las permanentes. Una wallet sin allowance viva no está expuesta a esto.

El plan escalonado. La interfaz puede estrecharlo ya: topes de token no nulos obligatorios cuando un agente tiene acceso a protocolos, revocación de las allowances existentes al autorizar, y una pantalla de revocación de emergencia. Lo que lo cierra de verdad es la capa de validators: un validator profundo con oráculo o TWAP que rechace un swap cuya salida sea inverosímil frente al precio de mercado. Un vault o una contabilidad interna lo cerraría del todo y es la opción de horizonte más largo. La primera fase es trabajo de interfaz y se deja **fuera** del contrato a propósito — ver el presupuesto de bytecode en §5.

Limitaciones aceptadas

- **BVCC-09** — un solo guardián puede bloquear una recuperación. Denegación de servicio contra la vía de rescate, no robo. Elige los guardianes en consecuencia: el mecanismo asume que al menos dos de los tres son honestos y localizables.
- **BVCC-10** — pausar un agente no retira sus allowances. Revócalas explícitamente; pausar por sí solo no es contención.
- **BVCC-13** — faltan comprobaciones de dirección cero en los constructores de los validators. Inerte tal como está desplegado, verificado en las seis redes.

Recomendaciones operativas

1. **Migrar.** Una wallet desplegada no se actualiza en su sitio. Los fondos que se queden en V1, V2 o V3 corren sobre bytecode con BVCC-01 sin corregir.
2. **Autorizar agentes con una lista de destinatarios no vacía** que excluya la dirección del propio agente. En wallets sin migrar es lo que cierra BVCC-01; en V4 sigue siendo buena práctica.
3. **Poner a cero las allowances vivas** antes de autorizar un agente — la mitigación de BVCC-03, y la única de esta lista que V4 no vuelve redundante.
4. **Verificar los guardianes de una wallet en cada red antes de fondearla ahí.** Redundante en V4, sigue siendo necesario para wallets creadas por una factory anterior.
5. **Desplegar proactivamente en todas las redes soportadas** para no dejar ninguna dirección sin reclamar.

V4 hace innecesarias tres de las cuatro mitigaciones provisionales que llevaba este informe: la apropiación de guardianes se cierra en la factory, la credencial la emite la wallet y una recuperación pausa los agentes. Sobrevive una, y no es cosa de contratos: **poner a cero cualquier allowance viva hacia los protocolos que un agente pueda alcanzar antes de autorizarlo**. Esa es la exposición del caso 3 de BVCC-03, y sigue abierta hasta que llegue la fase de validators.

Quien esté en V1, V2 o V3 sigue sobre el bytecode viejo hasta que migre. Una wallet desplegada no se puede actualizar en su sitio, así que todos los hallazgos anteriores siguen aplicando a los fondos que se queden atrás —incluida la reentrada cruzada de BVCC-01, explotable en las tres generaciones previas—. La interfaz detecta una wallet obsoleta por su dominio EIP-712 y propone la migración; el `kill()` de las factories sustituidas solo se hace cuando los usuarios se hayan movido.

7 Conclusión y limitaciones

La BVCC Agent Wallet aplica un modelo de permisos completo y por capas. Se identificaron tres incidencias de severidad alta a lo largo de tres rondas de revisión, cada una reproducida, remediada, redesplegada y re-verificada, y la batería de permisos on-chain pasa sobre el bytecode parcheado. Al preparar esta edición aparecieron cuatro más: una reentrada cruzada, la apropiación de guardianes por la factory, el gasto sin acotar del caso 3 (BVCC-03) y los agentes sobreviviendo a una recuperación. Todas menos una están corregidas y desplegadas en la generación V4 de factories, junto con los arreglos de la credencial, la rotación de guardianes y el `approve` con valor. La recuperación por guardianes se ejecutó y verificó de extremo a extremo — rotación de propiedad a una nueva passkey y rechazo on-chain de la clave antigua (ver Anexo D). La ronda V2 corrigió un problema de gas en mainnet en el snapshot de comisiones y concluyó con un despliegue multichain determinista (ver Anexo D).

La ronda V3 es el cambio más profundo hasta ahora en la vía del agente. Cerró un vector de exfiltración que los presupuestos no podían ver, haciendo las llamadas DeFi del agente default-deny por selector y anclando el destino o bien a la propia wallet o bien a un validator on-chain cuyo registro es asimétrico: inmediato para denegar, 48 horas para permitir (ver Anexo A y Anexo A). Dentro de ese trabajo se cazaron dos bypasses en potencia: una máscara de comandos que habría aceptado un depósito en un puente cross-chain como si fuera un swap, y una vía `increaseLiquidity` sin comprobación de propiedad, que se dejó deliberadamente sin registrar. El modelo está verificado por 264 tests unitarios, de fork y de fuzz, por un swap real en mainnet en el que todos los destinatarios resuelven a la wallet y la comisión es exacta, y por el rechazo de la variante adversarial de ese mismo swap antes de ejecutarse (ver Anexo B).

V4 cerró esa ronda y está vivo en las seis redes, con la capa de validación profunda completa: los dos selectores que delegan en el registry —el `execute` del Universal Router y el `modifyLiquidities` del PositionManager v4— tienen validator activo en todas partes, y el ciclo de liquidez completo se ejecutó con un agente en mainnet de Arbitrum, terminando con la posición quemada y sin valor nativo varado en el PositionManager (ver Anexo A y Anexo D).

El riesgo residual se expone sin adornos. Las llamadas a protocolo (caso 3) son seguras en el sentido que V3 se propuso: el destino queda anclado. Pero anclar el destino no es lo mismo que acotar el valor, y donde exista una allowance viva esa diferencia es la pérdida entera. Ese hueco sobrevive a V4: una clave de agente comprometida sobre una wallet dejada en el default permisivo, con una allowance ya concedida, sigue valiendo más que su presupuesto. El plan escalonado del BVCC-03 dice qué puede estrechar la interfaz ahora y qué necesita la capa de validators para cerrarse de verdad. Aparte, todos los hallazgos de este informe siguen aplicando a los fondos que estén en wallets V1, V2 o V3, porque una wallet desplegada no se puede actualizar en su sitio: la exposición termina cuando el usuario migra, no cuando sale V4.

Qué establece y qué no establece esta revisión

Establece que los hallazgos concretos aquí recogidos se reprodujeron, se remediaron donde así se indica, y que la remediación está presente en bytecode desplegado en las redes del Anexo F — cada una releída de la cadena, no deducida de un script de despliegue. Cada hallazgo tiene su test de regresión y la suite pasa.

No establece que los contratos estén libres de otros defectos. Es una revisión interna hecha por el equipo que escribió el código. Los cuatro hallazgos aportados por una pasada posterior con otra herramienta —BVCC-04, BVCC-06, BVCC-07 y BVCC-11— habían sobrevivido cada uno a tres rondas anteriores, que es la medida más clara disponible de lo que se deja repetir el mismo método.

Ningún tercero independiente ha revisado este código, así que nada de lo que hay aquí lo ha comprobado alguien con interés en encontrar fallos. Los contratos no han sido auditados externamente y este documento no debe leerse como sustituto de una auditoría.

La cobertura tampoco es uniforme por red. La batería de permisos se ejecutó en Arbitrum Sepolia; la capa de call policies y el ciclo de liquidez de Uniswap v4, en Arbitrum One. En las otras cuatro redes los contratos están desplegados y verificados byte a byte, y se releyó el cableado de los validators, pero no se ejecutó ningún ciclo de extremo a extremo.

A Arquitectura de seguridad

Material de referencia sobre los mecanismos a los que aluden los hallazgos: cómo se valida una llamada de agente, qué codifica una call policy y cómo funciona el registry de validators on-chain.

Modelo de ejecución del agente

El agente llama a `executeAsAgent(bytes32 mode, bytes executionData)` como una transacción EOA ordinaria (no una UserOp). `mode` es el selector de batch ERC-7579 (`0x0100…00`); `executionData` es un `Execution[]` codificado en ABI de `(target, value, callData)`. El orden de validación dentro de la llamada es **per-tx → destinatario/protocolo → período → diario → total**, con el estado actualizado **antes** de la ejecución externa (checks-effects-interactions) y un guard `nonReentrant`.

ALTA (exfiltración de fondos por el agente) **CORREGIDO — V3 DESPLEGADO 2026-07**

Descripción

Hasta V2, el presupuesto del agente se descontaba cuando el valor salía de la wallet por una vía que la wallet podía ver: una transferencia nativa, un `transfer` ERC-20 o un `approve`. Una llamada a protocolo (caso 3) se permitía en cuanto su destino estaba en `allowedProtocols`. Eso no basta, porque varios puntos de entrada DeFi reciben el destino como argumento:

- `Pool.withdraw(asset, amount, to)` de Aave envía la posición suplida directamente a `to` — sin `approve` y sin `transfer`, así que el presupuesto no se enteraba.
- Las llamadas de swap y LP (`exactInputSingle`, `execute` del Universal Router, `mint`) llevan un argumento `recipient` que el agente elegía libremente.

Una clave de agente robada o defectuosa podía por tanto poner su propia dirección como destino y vaciar todo lo que la wallet hubiera suplido o aprobado a un protocolo whitelisteado **sin consumir un solo wei de su presupuesto**. La severidad es ALTA: la pérdida solo está acotada por la exposición DeFi de la wallet, y la contabilidad on-chain no muestra nada.

Remediación (V3) — call policies por selector

Las llamadas de caso 3 son ahora **default-deny por selector**. Whitelistar el protocolo ya no basta: el propietario debe registrar además una policy que diga *a dónde puede ir el dinero*. Las policies se escriben con `setCallPolicy(target, selector, policy)`, que solo puede llamar la propia wallet — en la práctica el passkey del propietario, incluido en la misma firma que autoriza al agente. La policy es una única palabra `uint256`:

BITS	CAMPO	SIGNIFICADO
255	`POLICY_ALLOWED`	El selector está permitido. Sin él → `SelectorNotAllowed`
254	`POLICY_DEEP`	Deriva la llamada al registry de validators on-chain (ver Anexo A)
223..192	`PIN_WALLET` (bitmap 32 bits)	La palabra *i* del calldata debe ser `address(this)`
191..160	`PIN_PROTOCOL` (bitmap 32 bits)	La palabra *i* debe ser una dirección de `allowedProtocols`

La comprobación dentro de `executeAsAgent` ocurre antes de cualquier llamada externa: (1) se lee el selector del calldata (`< 4` bytes → `CalldataTooShort`); (2) la policy debe llevar `POLICY_ALLOWED`; (3) cada palabra anclada se compara **a palabra completa**, lo que además rechaza bits altos sucios, y una palabra leída más allá del final del calldata vale cero y por tanto falla — la comprobación es fail-

closed por construcción; (4) si está `POLICY_DEEP`, el registry debe devolver `true`. Se añadieron cuatro custom errors: `SelectorNotAllowed`, `PinnedArgMismatch`, `CalldataTooShort` y `PolicyValidationFailed`. También cambió un default relacionado: una lista `allowedProtocols` vacía ahora revierte `NoProtocolsWhitelisted` en lugar de significar «cualquier protocolo».

> **El propietario nunca está sujeto a las call policies.** Solo aplican a los agentes. Una persona que firma con su passkey sigue usando cualquier dApp con normalidad — V3 estrecha la vía delegada, no la del propietario.

Policies registradas (se muestra Arbitrum One; las otras cinco redes usan las mismas formas)

PROTOCOLO	SELECTORES	ANCLAJE
Uniswap SwapRouter02	`exactInputSingle` · `exactOutputSingle`	PIN_WALLET palabra 3
Uniswap SwapRouter02	`exactInput` · `exactOutput` (multi-hop)	PIN_WALLET palabra 2
Aave v3 Pool	`supply` · `withdraw` · `borrow` · `repay`	PIN_WALLET palabra 2 / 2 / 4 / 3
Aave v3 Pool	`repayWithATokens` · `setUserUseReserveAsCollateral` · `setUserEMode`	Permitido sin anclaje — no existe argumento de destino
Permit2	`approve(address,address,uint160,uint48)`	PIN_PROTOCOL palabra 1 (el spender)
Uniswap Universal Router	`execute(bytes,bytes[],uint256)`	DEEP → registry de validators
Uniswap v3 NFPM	`mint` · `collect`	PIN_WALLET palabra 9 / 1
Uniswap v3 NFPM	`decreaseLiquidity` · `burn`	Permitido sin anclaje — el NFPM los limita al propietario del `tokenId`
WETH	`deposit` · `withdraw`	Permitido sin anclaje — ambos acreditan/pagan a `msg.sender`

> **Dos trampas de calldata encontradas al escribir los presets.**
>
> - **Desplazamiento del offset.** El struct de `exactInput` (multi-hop) lleva un `bytes path`, lo que vuelve dinámica la tupla y mete una palabra de offset por delante: el destinatario está en la palabra 2, no en la 3 como en las variantes `*Single`. Anclar ahí la palabra 3 habría anclado `amountIn` y habría dejado el destino libre. Todos los offsets se re-derivaron codificando calldata real.
> - **`increaseLiquidity` queda deliberadamente fuera de la whitelist.** A diferencia de `decreaseLiquidity`, `collect` y `burn`, el NFPM de v3 no le aplica ninguna comprobación de propiedad — solo el deadline. Un agente habría podido empujar los tokens que la wallet tiene aprobados al NFPM hacia un `tokenId` propiedad del atacante: un bypass de la lista de destinatarios que ningún anclaje de bitmap puede expresar, porque la palabra 0 es un `tokenId`, no una dirección. Los agentes añaden liquidez minteando una posición nueva con

el destinatario anclado a la wallet; reactivar `increaseLiquidity` exigiría un validator profundo que compruebe `ownerOf(tokenId) == wallet`.

Qué cubre cada capa

CASO	OPERACIÓN	GOBERNADO POR
1	Transferencia de ETH nativo	`allowedRecipients` + límites nativos
2	`transfer(to, amount)`	`allowedRecipients` + límites de token
2b	`approve(spender, amount)`	`allowedRecipients` + límites de token
3	Llamada DeFi / a protocolo	`allowedProtocols` + **call policies**

Nota de alcance. V3 cierra la exfiltración de fondos por swaps, LP y retiradas de protocolo: el caso 3 es ahora seguro por construcción, configure lo que configure el propietario. Lo que no hace es convertir a un agente comprometido en inofensivo — un `transfer` o un `approve` directos (casos 2 y 2b) siguen gobernados por la lista de destinatarios y los límites por token, y esos los elige el propietario. Simulado en mainnet contra el agente de pruebas, con la lista de destinatarios vacía y los límites a cero, un `USDC.transfer` directo a un atacante pasa; con destinatarios y límites configurados revierte `RecipientNotAllowed`. Los límites on-chain son el modelo de seguridad.

Por qué hace falta un segundo mecanismo

Un anclaje de bitmap solo alcanza una palabra en un offset fijo. En el `execute(bytes commands, bytes[] inputs, uint256 deadline)` del Universal Router — y en el `modifyLiquidities` de Uniswap v4 — el destinatario va enterrado dentro de datos de longitud variable, así que no hay offset fijo. Esos selectores llevan en su lugar el bit `DEEP`, que delega la decisión en un validator on-chain sin estado.

Diseño

- **Dirección de despacho fija.** `BVCCValidatorRegistry` está compilado dentro de la wallet como constante (`0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2`). Como la palabra de policy solo lleva un flag y nunca una dirección, a un propietario no se le puede colar el validator de un atacante. Un test dedicado (`Create2Consistency.t.sol`) rompe la build si esa constante llega a desviarse de la dirección CREATE2 del registry.
- **Despacho fail-closed.** Sin validator registrado para el destino → `false`. Un validator que revierte, o que no tiene código, también deniega. Solo un `true` explícito deja pasar la llamada; cualquier otra cosa revierte `PolicyValidationFailed`.
- **Validators sin estado.** Son contratos `view` a los que se llega por `staticcall`. Nunca custodian fondos, nunca reciben approvals y no pueden mutar estado — solo responden sí o no.
- **Doble consentimiento.** Habilitar un protocolo complejo exige dos pasos independientes: BVCC registra el validator en el registry y el propietario registra la policy `ALLOWED|DEEP` con su passkey. Si falta cualquiera de los dos, la llamada se deniega.

Gobernanza asimétrica

DIRECCIÓN	FUNCIÓN	DEMORA
Permitir — registrar o sustituir un validator	`proposeValidator` → `activateValidator`	**Timelock de 48 h** + evento `ValidatorProposed`

Denegar — cancelar una propuesta pendiente	`cancelProposal`	Inmediata
Denegar — desactivar un validator activo	`freezeValidator`	Inmediata

La demora solo corre en la dirección permisiva, así que una propuesta maliciosa o equivocada es pública durante dos días antes de poder surtir efecto y los propietarios pueden reaccionar (pausar o revocar sus agentes). El peor caso con la clave de administración robada es por tanto una denegación de servicio o —tras 48 horas visibles públicamente— una degradación al nivel de protección de V2 en un protocolo. Nunca es una vía directa a los fondos: el registry no mueve nada.

El validator del Universal Router

- **Atado a un único router.** La dirección del router es un argumento inmutable del constructor; una llamada con cualquier otro destino devuelve `false`. Un router nuevo necesita su propio validator y su propio ciclo de gobernanza de 48 horas.
- **Coincidencia exacta del byte de comando** sobre `{0x00 V3_SWAP_EXACT_IN, 0x01 V3_SWAP_EXACT_OUT, 0x10 V4_SWAP}`. Cualquier otro byte se deniega, incluidas las variantes allow-revert y los comandos desconocidos.
- **Se comprueba cada destinatario**, incluida la acción `TAKE` de v4 (`{0x07 SWAP, 0x0b SETTLE, 0x0e TAKE}`): debe ser la wallet o el centinela `MSG_SENDER`. `SWEEP`, `TRANSFER`, `PAY_PORTION`, `WRAP` y `UNWRAP` se deniegan de plano.
- **El input malformado deniega.** Un calldata indecodificable revierte dentro del validator, y el registry lo trata como denegación; una cadena `commands` vacía se rechaza explícitamente.
- **Alcance.** Swaps token→token. Las patas nativas que dependen del `ADDRESS_THIS` del router quedan deliberadamente fuera; los swaps nativos se ejecutan como batches propios de BVCC a través de WETH, de modo que el router nunca retiene fondos de la wallet a mitad de camino.

> **Bug crítico detectado en revisión — la máscara de comandos.** La primera versión del validator enmascaraba los bytes de comando con `& 0x3f`. El router real enmascara con `& 0x7f` (el bit `0x80` es `FLAG_ALLOW_REVERT`), lo que significa que `0x40` **no** es un alias de `0x00`: es `ACROSS_V4_DEPOSIT_V3`, un depósito en un puente cross-chain. Con la máscara equivocada el validator lo habría aceptado como un swap v3 mientras el router bridgeaba los fondos de la wallet hacia un atacante en otra cadena: exactamente la clase de exfiltración que V3 existe para impedir. La corrección sustituyó la máscara por coincidencia exacta, ató el validator a un único router y rechazó las cadenas de comandos vacías. El comportamiento se fuzzea ahora sobre los 256 bytes de comando contra un modelo del router, y se volvió a confirmar en mainnet tras el despliegue (ver Anexo B).

Aave no necesita validator

Los argumentos de destino de Aave (`onBehalfOf`, `to`) están en offsets fijos, así que se resuelven con anclajes de bitmap dentro de la propia wallet. Ninguna policy de Aave lleva el bit `DEEP`, el registry no se consulta nunca para Aave y no interviene ningún timelock. Solo necesitarían validator las operaciones cuya seguridad no se puede expresar como «la palabra *N* debe ser la wallet» — flash loans, multicall, rutas por adaptadores — y esas quedan fuera de alcance.

Validators registrados — completo en todas las redes

Exactamente **dos** selectores llevan el bit `DEEP` y por tanto consultan el registry: `execute(bytes,bytes[],uint256)` del Universal Router y `modifyLiquidities(bytes,uint256)` del PositionManager v4 de Uniswap. Los dos están registrados y activos en las seis redes, leídos de `validators(address)` en el registry:

RED	VALIDATOR DEL UNIVERSALROUTER	VALIDATOR DEL POSITIONMANAGER V4
Arbitrum One	`0xD1522594e185C882bfDEc6FFD4DE8a53F69AF0Ca`	`0x6aE9c36121D5cD2d4Bde816F7F857AfB36Ccb8Eb`
Ethereum	`0x3615a0Fc0663C0201B543deC9816d5Ef30a82ffb`	`0xeFd47Aeb41e532D6a4ed04553b25827d88a4364d`
BNB Chain	`0x6945f861e0D7FCD566739000f3fDC1D44896D815`	`0x5f5Ee1068c2c54bBE896A08444958a289407Cd35`
Polygon	`0xC1C584E4149eD2EFe20DA0155b1B8257232102fF`	`0x62Eb47697BcF63dC9450B662d259AFf555958e25`
Base	`0x4A2Fc73AAdEfC84513defaa0c444d2150AC6A6D8`	`0xa2F35323DAcBFC2b5909B15607b511c6953e78a5`
Arbitrum Sepolia	`0x87550034627966e32d82E3b8AdB3f19c62E8d86E`	`0x32aA1B3E35dc6fF09EB7CDc8F6b43F3573AD2182`

Cada validator de PositionManager se comprobó uno a uno contra el PositionManager v4 al que está atado por su inmutable `POSITION_MANAGER`, y todos resuelven a la dirección correcta de su cadena. Los seis comparten el mismo `HOOK_REGISTRY` (`0x551C6e7ABdA04a110790888e711198f25621b066`, CREATE2, con código presente en cada una). Un atado cruzado habría fallado en cerrado y denegado en silencio toda la operativa de liquidez; de ahí que se verifique por red en lugar de deducirlo del script de despliegue.

> El `NonfungiblePositionManager` de Uniswap **v3 no** tiene validator registrado en ninguna red, y eso es correcto, no es un hueco. Sus policies son anclajes de bitmap — `mint` ancla el destinatario en la palabra 9, `collect` en la 1, y `decreaseLiquidity` y `burn` los controla el propio NFPM por dueño —, así que ninguna policy de v3 lleva el bit `DEEP` y el registry no se consulta jamás. `BVCCPositionManagerValidator` es un contrato de v4: está atado a un único PositionManager v4 e interpreta bytes de acción de v4. Registrarlo contra el NFPM de v3 rompería la liquidez v3 en lugar de asegurarla. `increaseLiquidity` sigue deliberadamente sin registrar por lo dicho en Anexo A: el NFPM no comprueba el dueño en esa función.

B Evidencia de pruebas

Las baterías de pruebas detrás de los hallazgos y sus remediaciones. Cada hallazgo del §4 tiene su test de regresión; estas son las baterías que ejercitan el modelo en conjunto.

Ejecutada contra la wallet parcheada `0x1ED261…F0C3` con el EOA del agente `0x3852…4aB4`. Reverts confirmados por simulación; casos válidos como transacciones reales. Los destinatarios permitidos fueron los dos guardianes de recovery (controlados por el bot, fondos recuperables).

#	PRUEBA	ESPERADO	RESULTADO
1	ETH per-tx (0,0002 > máx 0,0001)	revierte	ExceedsPerTxLimit
2	ETH dentro de límite (0,0001)	pasa	Success
3	USDC per-tx (2 > máx 1)	revierte	ExceedsTokenMaxAmount
4	USDC diario (1+1+1, cap 2) — txs reales	la 3ª revierte	TokenDailyLimitExceeded
5	Envío de token a destino no permitido	revierte	RecipientNotAllowed
6	Envío de ETH a destino no permitido	revierte	RecipientNotAllowed
7	Presupuesto total (0,0002+0,0001 = 0,0003) — txs reales	la siguiente revierte	AgentBudgetExceeded
8	Presupuesto período (0,0003+0,0002 = 0,0005) — txs reales	la siguiente revierte	PeriodBudgetExceeded
9	Auto-rollover del período (periodStart 0)	la 1ª tx resetea la ventana	Verificado
10	Pausado (el owner pausa)	agente bloqueado	EnforcedPause
11	Expiración (caducó sola)	agente bloqueado	AgentPermissionsExpired

> **Observación.** La lista blanca de destinatarios aplica también a los **transfers de token**, no solo a ETH — un envío de token a una dirección no permitida revierte `RecipientNotAllowed`. La re-autorización vía “Edit” no resetea el contador diario (indexado por día UTC); con `periodStart = 0` la primera operación rota la ventana del período.

Resistencia a escalada de privilegios (de la sesión del 06-06)

Un agente que apunta a la propia wallet para llamar a funciones de owner (`pauseAgents`, `revokeAgent`, `authorizeAgent`) revierte con `AgentCannotCallWallet`; las llamadas arbitrarias a contratos fuera de la lista de protocolos están bloqueadas. El agente solo puede mover ETH y tokens permitidos hacia destinatarios permitidos.

Batería de pruebas — `forge test`: 264 pasan, 0 fallan

SUITE	TESTS	FOCO
`BVCCAgentWallet.t.sol`	50	Núcleo de permisos del agente: límites, presupuestos, expiración, pausa, batches

`UniversalRouterValidator.t.sol`	42	Corpus del router, destinatarios atacantes, fuzz sobre los 256 bytes de comando
`BVCCAgentWalletAdvanced.t.sol`	40	Contabilidad de comisiones, regresiones de `approve`, casos límite
`CallPolicyAdversarial.t.sol`	34	Intentos de bypass de policies, gobernanza del registry, calldata malformado
`BVCCPositionManagerValidator.t.sol`	26	Validación del calldata de LP en Uniswap v4
`BVCCWallet.t.sol`	23	Smart account base: ejecución, comisiones, recuperación por guardianes
`BVCCWalletFactory.t.sol` · `BVCCAgentWalletFactory.t.sol`	18	Factories CREATE2, direcciones deterministas, kill switch
`BVCCHookRegistry.t.sol`	11	Registry de aprobación de hooks y su timelock
`AaveIntegration.t.sol`	10	Ciclo de Aave sobre fork de mainnet y variantes adversariales
`BVCCPMValidatorSdkVectors.t.sol` · `ERC7739*.t.sol` · `Create2Consistency.t.sol` · `GasBench.t.sol`	10	Vectores dorados del SDK, firmas ERC-7739, guarda de congelación de direcciones, gas

Tests adversariales de policies (`CallPolicyAdversarial.t.sol`): los `withdraw`/`supply`/`borrow`/`repay` de Aave con un `to` u `onBehalfOf` externo revierten `PinnedArgMismatch`; un selector no registrado revierte `SelectorNotAllowed`; `PIN_PROTOCOL` rechaza un spender no whitelisteado; una llamada DEEP deniega cuando el validator devuelve false, revierte, no tiene código o no está registrado, y solo pasa con `true`; el timelock de 48 h, el freeze y el cancel se comportan según lo especificado; `policy = 0` revoca; el calldata corto, los anclajes fuera de rango y los bits altos sucios se rechazan; y un batch con un solo elemento malo revierte entero.

Aave sobre fork de mainnet (`AaveIntegration.t.sol`): ciclo completo supply → borrow → repay → withdraw, con la comisión de agente del 0,15% cobrada exactamente en `withdraw` y `borrow` y no cobrada en `supply` ni `repay`; consumo de presupuesto verificado por la vía del `approve`; un beneficiario externo en cualquiera de las cuatro llamadas revierte; `flashLoanSimple` y un `setUserEMode` no registrado revierten; un Pool ausente de `allowedRecipients` revierte; y el batch approve+supply hace rollback atómico si falla.

La capa de permisos se había atacado desde todos los ángulos a lo largo de tres rondas; la recuperación por guardianes y la vía de firma, no. Las dos custodian fondos, así que las dos recibieron el mismo trato: veinte tests escritos para conseguir que funcione algo que no debe (`test/RecoveryAdversarial.t.sol`, `test/SignatureAdversarial.t.sol`).

14.1 Vía de firma — sin hallazgos

Diez ataques contra la vía ERC-7739 / WebAuthn, los diez rechazados. Los dos que más importan son los de replay, porque el diseño de esta wallet invita a ambos: la dirección es idéntica en todas las redes, y una misma passkey respalda una smart wallet y una agent wallet en dos direcciones distintas.

ATAQUE	RESULTADO

Reutilizar en otra red una firma hecha para la misma dirección	**Rechazado** el chainId va atado en el dominio del verificador
Reutilizar una firma entre la smart wallet y la agent wallet del mismo dueño	**Rechazado** el contrato verificador va atado
Firma maleada (`s` sustituida por `N − s`)	**Rechazado**
Firma a cero	**Rechazado**
Firma sobre otro contenido, o sobre el dominio de otra dApp	**Rechazado**
`authenticatorData` manipulado (quitando el flag de verificación de usuario)	**Rechazado**
Challenge cambiado por otro digest	**Rechazado**
`webauthn.create` colado donde se exige `webauthn.get`	**Rechazado**

Verificación on-chain — Arbitrum One (mainnet)

Wallet de pruebas `0x605A4C65dBa64bd26d9F7701e1e4Cda48c594A5c`, agente `0x38529C66F3cf22453D66B9E2A20FdF2676544aB4`.

FECHA	OPERACIÓN	RESULTADO
2026-07-16	Swap del agente USDC→WETH por SwapRouter02, dirigido por el servidor MCP `0x4768603d38b03f9823d5d25e53fb03970b8e4d294ebee791d0ae9ef92e71a8ae`	**OK** USDC debitado exacto, WETH a la wallet, comisión 0,15% exacta
2026-07-20	Swap del agente por el Universal Router — batch atómico de 3 llamadas que atraviesa tres capas distintas `0x6f831b37ae5db240256bf6bf99c8ce418edb75a1c38e37d6ed3e2c3a052a3bc0`	**OK** gas 419.124 · USDC −0,100000 · comisión 0,000000078468831120 WETH = 0,15% exacto · todos los destinatarios, la wallet
2026-07-20	El mismo swap con `recipient` = atacante (simulado, sin mover fondos)	**DENEGADO** revierte antes de ejecutar

Las tres llamadas de ese batch atraviesan cada una un guardián distinto: `USDC.approve(Permit2)` es caso 2b y se comprueba contra `allowedRecipients`; `Permit2.approve` es caso 3 con `PIN_PROTOCOL` en el spender; `UniversalRouter.execute` es caso 3 con `DEEP` y lo decide el validator.

Comportamiento del gate, medido antes y después de la activación del validator (48 h):

REGISTRY.VALIDATE(...)	ANTES DE ACTIVAR	DESPUÉS DE ACTIVAR
Swap legítimo (recipient = wallet)	false	**true**
Intento de robo (recipient = atacante)	false	false
Comando `0x40` (puente cross-chain)	false	false
Destino ≠ el router al que está atado	false	false

La corrección de la máscara queda por tanto confirmada en mainnet, no solo en tests, y la propiedad fail-closed se ve en la columna «antes»: hasta que el validator no estuvo activo, incluso el swap legítimo se denegaba.

Gas

OPERACIÓN	GAS
`createWallet`	`4.548.977`
`authorizeAgent`	`166.803`
`setCallPolicy`	`25.912`
`executeAsAgent` (caso 3, con policy comprobada)	`64.758`

C Análisis estático

Slither se pasó sobre el árbol en cada ronda. Este anexo recoge los resultados y, tan importante como eso, qué resultados son falsos positivos y por qué — para no volver a investigarlos, y para que nadie borre las dos funciones virtuales que marca como código muerto.

Slither 0.11.3 se ejecutó sobre ambos repositorios antes del redespliegue. Reportó 39 resultados en 59 contratos; **ninguno accionable**. La tabla resume las categorías y su tratamiento.

CATEGORÍA	EJEMPLOS	TRATAMIENTO
Patrones por diseño	Comparaciones con `block.timestamp`, ensamblador, llamadas de bajo nivel, calls-en-bucle (batch)	Aceptado
Falsos positivos	Mapping “no inicializado”; reentrancy en `createWallet` (solo un `emit` tras un `setGuardians` de confianza)	Sin acción
Código muerto — `_feeNumerator()`	Marcado como no usado	Falso positivo — verificado en uso (ver nota)
Falta zero-check — `owner_` de la factory	El constructor asigna owner sin validar	Corregido
Cosmético	Nomenclatura, dígitos literales, complejidad ciclomática	Aceptado

> **Nota de verificación — un hallazgo de herramienta no es un hecho.** Slither marcó `_feeNumerator()` como código muerto eliminable. Una comprobación del código mostró que **sí** se usa para calcular la comisión (base 0,05% / override del agente 0,15%); eliminarlo habría roto la lógica de comisiones. El hallazgo era un falso positivo causado por el dispatch de funciones virtuales y **no** se actuó sobre él.

Remediación aplicada

Se añadió un control defensivo de dirección cero — `if (owner_ == address(0)) revert ZeroOwner();` — al constructor de ambas factories, sincronizado en los dos repositorios. Suite completa en verde: **forge test = 128/128** en cada repo. Slither se re-ejecutó sobre los contratos V2 tras el endurecimiento de gas (Anexo D): **0 resultados**; la suite unitaria queda en **131/131**.

Análisis estático — Slither re-ejecutado sobre V3

Slither 0.11.3 se volvió a pasar sobre el árbol V3 el 2026-07-27 (`slither . --exclude-dependencies`, 65 contratos, 100 detectores): **54 resultados — 3 High, 29 Low, 22 Informational**. Cada resultado de impacto alto se contrastó contra el código fuente antes de aceptarlo o descartarlo. Uno de ellos es real: el resultado `reentrancy-eth` es un hallazgo ALTO confirmado, reproducido con un test de prueba de concepto y todavía abierto a fecha de redacción (BVCC-01).

> **La pasada que cuenta es esta, no la de V2.** La de V2 devolvió «0 resultados», pero solo porque el analizador no pudo bajar a IR el `ReentrancyGuard` con transient storage de OpenZeppelin y abandonó (Anexo C). Aquel cero era la herramienta rindiéndose, no un aprobado. El árbol de V3 se analiza de principio a fin —los detectores sí llegan a

> `executeAsAgent`, como demuestra el resultado de reentrancy de abajo—, así que este es el dato de análisis estático que tiene peso, y sus clases de resultados encajan con los 39 de la pasada de V1.

DETECTOR	OBJETIVO	VEREDICTO
`reentrancy-eth` confianza: media	`executeAsAgent` escribe `_currentAgent = 0` después de `super.execute()`	**CONFIRMADO — ALTA** — reproducido con un test PoC (BVCC-01)
`uninitialized-state` ×2 confianza: alta	`_tokenTotalSpent`, `_tokenDailySpent`	**Falso positivo** — ambos son `mapping`, que no tienen inicializador y valen cero por defecto

D Previamente remediado — V1 y V2

Estos hallazgos se reprodujeron y cerraron en generaciones que V4 sustituye. Se conservan con su evidencia porque las wallets creadas por aquellas factories siguen vivas hasta que sus dueños migren, y porque V4 hereda las remediaciones que describen.

ALTA CORREGIDO Y REDESPLEGADO

Descripción

Los presupuestos **diario** y **total** por token del agente podían saltarse por completo. La validación contabilizaba el gasto de `transfer`, pero cargaba `approve` solo contra el cap por transacción — no contra el diario ni el total, y sin restringir el spender. El agente podía por tanto `approve` un allowance (solo capeado por per-tx) y vaciarlo con un `transferFrom` externo, sin tocar nunca los contadores acumulados.

Prueba de concepto (ejecutada on-chain, presupuestos agotados a diario 2/2 · total 2/3)

PASO	ACCIÓN	RESULTADO
1	`executeAsAgent → USDC.approve(B, 1)`	Pasa — per-tx OK, diario ignorado
2	Como EOA: `USDC.transferFrom(wallet, B, 1)` (fuera de `executeAsAgent`)	1 USDC extraído; contadores sin cambios

Un `transfer` plano del mismo importe sí revirtió correctamente con `TokenDailyLimitExceeded`; `approve` no. Al no tener límite diario, el ciclo `approve(1) → transferFrom(1)` era repetible — vaciando la wallet más allá de los caps diario (2) y total (3). Los fondos de la PoC se devolvieron a la wallet.

Remediación

En el bucle de acumulación de `executeAsAgent` se extendió la condición con `|| _isApprove(batch[i].callData)` de modo que `_applyTokenSpend` carga los `approve` contra los mismos caps diario/total y actualiza los contadores. La elección es deliberadamente conservadora (un `approve` sobrescribe el allowance, por lo que puede sobre-contar — la dirección segura). Se añadieron seis tests de regresión; **5 fallan sin el fix, todos pasan con él.**

> **Superficie acotada.** Los tokens aceptan solo `transfer` y `approve` por selector; `transferFrom` e `increaseAllowance` están bloqueados. ETH no tiene primitiva de delegación, así que no se ve afectado. `approve` era el único vector — ahora cerrado y re-verificado tras el redespliegue (ver Anexo B).

Ambas factories cambiaron de bytecode (zero-check), produciendo nuevas direcciones deterministas CREATE2 en Arbitrum Sepolia. La configuración de red del frontend se actualizó en consecuencia.

CONTRATO	NUEVA DIRECCIÓN (PARCHEADA)	ANTERIOR (DEPRECADA)
AgentWalletFactory	`0xc87aa10747A92B472EF6B36e190B84c897a2953e`	`0xe7ad…7e04`
SmartWalletFactory	`0xa5290A51a73903176e09C864E1542a07da67BD12`	`0x6890…F58d`

Separación de privilegios

- **Deployer (sin control):** `0xA3e06B6443D911915bb4eD157832d43C25D6617D` — el `owner` de la factory se fija desde un argumento del constructor, no desde `msg.sender`, así que el deployer no tiene privilegios.
- **Admin / kill-switch:** `0x3145Bd5e2489d8bDdAb17F23F26F07Ac5aD55A4c` — inmutable; la única acción privilegiada es el `kill()` de un solo sentido.

Tamaños runtime dentro del límite EIP-170 (AgentWallet 21.169 B; AgentWalletFactory 23.803 B — 773 B de margen).

La wallet base admite recuperación social: un quórum de 2 de 3 guardianes puede rotar la passkey del propietario tras un timelock. Se probó de extremo a extremo en la wallet `0xEBc851bA9e8Ce32A2cb8eB99544F5F024f06E61d` y ya está **completada**.

Proceso

- **Inicio (2026-06-06).** Recovery propuesta hacia una nueva passkey de propietario; los guardianes 1 y 2 (`0xa337…5908`, `0x8e06…2CcC`) aprobaron — quórum 2/2.
- **Timelock.** `recoveryReadyAt` on-chain = 2026-06-08 17:24 UTC; durante toda la ventana el propietario conservó la capacidad de cancelar.
- **Ejecución (2026-06-08).** Transcurrido el timelock se llamó `executeRecovery()` desde la UI `/recover`. El `signer` on-chain rotó a la nueva passkey (X `0x303591…` / Y `0xb858ed…`), reemplazando al propietario anterior (X `0x070f7a…`).
- **Comprobación positiva.** El nuevo propietario firmó de inmediato una transferencia real de **19 USDC** con el nuevo Face ID — confirmada on-chain.

Prueba negativa — la passkey antigua queda anulada

Para confirmar que la rotación revocó realmente la clave anterior, se intentó una transferencia con la passkey **antigua**, cuya credencial WebAuthn sigue presente en el autenticador de pruebas. La firma se genera localmente, pero debe validarse on-chain contra el propietario actual.

#	ACCIÓN	ESPERADO	RESULTADO
1	Enviar 0.0001 ETH, UserOp firmado por la passkey ANTIGUA	rechazo en validación	FailedOp(0, AA24 signature error)
2	Efecto on-chain (saldos ETH / USDC y nonce)	sin cambios	Sin cambios

> **Resultado.** La recovery se completó con éxito: la propiedad pasó a la nueva passkey, la nueva clave opera la wallet y la antigua es rechazada criptográficamente por el EntryPoint durante la validación de firma — ya no puede mover fondos aunque su credencial local siga existiendo.

ALTA (disponibilidad / sobrepago) CORREGIDO — V2 DESPLEGADA 2026-06-11

Descripción

En Arbitrum mainnet, los swaps a través de la wallet fallaban en la estimación de gas. La lógica de comisiones del Caso 3 escanea el calldata buscando candidatos a token y sondea cada uno con `balanceOf`. Los precompiles del sistema de Arbitrum (`0x64–0x6f`) reportan bytecode (`0xfe`), por lo que el control por longitud de código no los filtraba — y llamarlos consume **todo el gas reenviado** en lugar de revertir barato. El estimador subía el límite de gas hasta ~32M para que la sonda “cupiera”; ese límite inflado es lo que la wallet habría prefinanciado (riesgo de sobrepago 64×) y, en la práctica, la operación fallaba.

	ANTES (BUG)	DESPUÉS (FIX V2)

Estimación de gas (frontend / bundler)	~32.000.000	~400.000
Efecto de un candidato quema-gas	quema todo el gas reenviado	capado a 100k, la ejecución continúa
Resultado	el swap falla (o sobrepago 64×)	swap normal, ~$0,02–0,05 en Arbitrum

Remediación (V2)

- **Sondas capadas en gas.** Una constante compartida `PROBE_GAS_CAP = 100_000` capa cada staticcall de `balanceOf` tanto en `_tryBalanceOf` (snapshot) como en `_collectFeesOnIncrease` (cobro de comisión). El peor caso de un escaneo completo queda acotado a ~2,2M de gas frente a los 32M anteriores; un ERC-20 real necesita mucho menos que el cap, así que la contabilidad de comisiones no se ve afectada.
- **El cobro de comisión nunca puede romper una llamada del usuario.** Si un token se comporta mal tras el swap, su comisión se omite en lugar de revertir la transacción del usuario.
- **Cota contable explícita.** `executeAsAgent` ahora exige `totalEth ≤ type(uint128).max` antes del cast, eliminando un caso límite de truncamiento silencioso para agentes configurados sin límites de ETH.

Verificación

- **forge test = 131/131** — tests de regresión añadidos: un candidato quema-gas no deja sin gas al swap; un `balanceOf` que revierte se filtra sin ser fatal; un token con retorno sobredimensionado se snapshotea igualmente.
- **Slither 0.11.3 re-ejecutado sobre V2: 0 resultados.** (Nota de herramienta: Slither no puede generar IR para los internos del `ReentrancyGuard` con transient storage de OpenZeppelin — limitación conocida del parser, no un hallazgo.)
- **Tamaños EIP-170:** AgentWallet 21.214 B; AgentWalletFactory 23.848 B (728 B de margen); SmartWallet 13.432 B; SmartWalletFactory 16.030 B.

Despliegue V2 — CREATE2 determinista, misma dirección en todas las redes

CONTRATO	DIRECCIÓN (IDÉNTICA EN TODAS LAS REDES)
SmartWalletFactory V2	`0x230b7010529AB6977Dd8581B3eF018ef865BdEf1`
AgentWalletFactory V2	`0x8D9e24022777173AD6336e00884b6C87c7EF054c`

Desplegadas el 2026-06-11 en Arbitrum One (42161), BNB Chain (56) y Arbitrum Sepolia (421614). Las wallets creadas por factories pre-V2 no reciben el fix — las wallets nuevas deben crearse a través de las factories V2.

Se corrigieron tres incidencias en la aplicación web de la wallet, sincronizadas con el repositorio público.

ÁREA	INCIDENCIA	CORRECCIÓN
`lib/wallet.ts`	La recuperación de credencial consultaba solo el evento de la factory base, por lo que reabrir una wallet *de agente* por dirección fallaba (“no hay wallet activa”).	Consultar también `agentFactory` / `AgentWalletCreated`.
Página de agentes	Los límites por token se mostraban solo como texto.	Leer `getTokenSpent` on-chain; pintar barras de progreso Diario/Total (verificado en vivo).
Flujo de deploy	El gas que pasaba la app infraestimaba el deploy de la agent wallet (~21 KB) en Arbitrum,	Delegar en la estimación de red de la wallet.

	provocando fallos al firmar.

E Cronología de la remediación

RONDA	FECHAS	FOCO	RESULTADO
V1	2026-06-06 → 06-08	Modelo de permisos del agente, primer análisis estático	BVCC-A1 encontrado, corregido y redesplegado; batería de permisos y recuperación por guardianes verificadas de extremo a extremo
V2	2026-06-11 → 06-12	Comportamiento del gas en mainnet	BVCC-A2 encontrado y corregido; despliegue multichain determinista en seis redes
V3	2026-07-21	Capa de call policies y registry de validators	Cerrada una vía de exfiltración de fondos; BVCC-02 y BVCC-12 cazados dentro del propio trabajo, antes de desplegar
V4	2026-07-27 → 07-28	Revisión posterior a la publicación, segunda pasada con otra herramienta	BVCC-01, BVCC-04 a BVCC-08 y BVCC-11 encontrados y corregidos; desplegado y verificado en seis redes
—	2026-07-28 → 07-29	Activación de validators	Capa de validación profunda completada en las seis redes; ciclo de liquidez v4 ejercitado en mainnet

A continuación, el registro de despliegue de V3, la build congelada, la capa off-chain y las notas de migración.

Despliegue V3 — CREATE2 determinista

CONTRATO	DIRECCIÓN (IDÉNTICA EN LAS SEIS REDES)
SmartWalletFactory V3	`0xD42F61AA856A4f47885Ecd2D0ce119411d53C192`
AgentWalletFactory V3	`0xd866a7563cDaC9F71423be3332b62c329C676064`
ValidatorRegistry	`0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2`
HookRegistry	`0x551C6e7ABdA04a110790888e711198f25621b066`

Redes: Arbitrum One (42161), Ethereum (1), BNB Chain (56), Base (8453), Polygon (137) y Arbitrum Sepolia (421614). Los cuatro contratos se re-comprobaron el 2026-07-27 vía la API de los exploradores: presentes, byte a byte idénticos y con código fuente verificado en las seis cadenas.

Los validators del Universal Router están atados a un router cada uno, así que su dirección cambia por red:

RED	VALIDATOR DEL UR	ESTADO EN EL REGISTRY
Arbitrum One	`0xD1522594e185C882bfDEc6FFD4DE8a53F69AF0Ca`	Activo
Ethereum	`0x3615a0Fc0663C0201B543deC9816d5Ef30a82ffb`	Activo
BNB Chain	`0x6945f861e0D7FCD566739000f3fDC1D44896D815`	Activo
Base	`0x4A2Fc73AAdEfC84513defaa0c444d2150AC6A6D8`	Activo
Polygon	`0xC1C584E4149eD2EFe20DA0155b1B8257232102fF`	Activo
Arbitrum Sepolia	`0x87550034627966e32d82E3b8AdB3f19c62E8d86E`	Activo

Estado leído directamente de `validators(router)` en cada cadena, la última vez el 2026-07-28, tras activar Arbitrum Sepolia. Un validator desplegado pero sin activar es inerte en la dirección segura: las llamadas DEEP correspondientes fallan cerradas. Los manifiestos de procedencia por red (direcciones más codehashes del router y del validator) están en `contracts/deployments/`. Los validators del `PositionManager` de Uniswap v4 se activaron en las seis redes en la misma tanda — ver Anexo A para las direcciones y la comprobación de atado por red, y Anexo D para el ciclo de liquidez ejecutado contra ellos.

Build congelada

Unas direcciones CREATE2 deterministas exigen una build congelada: solc **0.8.36**, `via_ir`, `optimizer_runs = 50`, EVM `cancun`. Ese compilador se eligió deliberadamente para esquivar el bug del pipeline via-IR del rango 0.8.28–0.8.33. El tamaño runtime de la factory de agentes es de **24.283 bytes**, con 293 bytes de margen bajo el límite EIP-170 — cualquier adición futura debe medirse contra ese margen.

Capa off-chain — SDK y MCP 0.2.0

Las librerías cliente se hicieron conscientes de las policies para que el guardián on-chain no se descubra solo al emitir la transacción. `getCapabilities` ahora condiciona un swap a las policies registradas *y* al registry de validators, no solo al struct de permisos, y devuelve la versión de la wallet junto con las policies que falten; los cuatro errores de V3 se decodifican en mensajes accionables que distinguen «no hay validator activo» (no reintentar) de «el validator rechazó este calldata» (corregible). `setCallPolicy` deliberadamente **no** se expone en el SDK: las policies las escribe el passkey del propietario, nunca la clave de un agente. El módulo de préstamos Aave y sus planners viajan por el mismo catálogo, siempre con el beneficiario anclado a la wallet.

Migración

El bytecode de la wallet V3 difiere del de V2, así que las direcciones CREATE2 también: los usuarios existentes crean una wallet nueva y mueven sus fondos, igual que en la migración V1→V2, tras lo cual se hace `kill()` de las factories V2. Las capacidades futuras — registrar más validators, añadir presets de protocolos — son neutrales respecto al bytecode y no obligarán a otra migración.

F Identificadores y direcciones

ELEMENTO	VALOR
Red	Arbitrum Sepolia (421614)
Wallet parcheada	`0x1ED261BA084c4E6117daf289f3ca233BBA3cF0C3`
Agente (rol B)	`0x38529C66F3cf22453D66B9E2A20FdF2676544aB4`
AgentWalletFactory (Sepolia, pre-V2)	`0xc87aa10747A92B472EF6B36e190B84c897a2953e`
SmartWalletFactory (Sepolia, pre-V2)	`0xa5290A51a73903176e09C864E1542a07da67BD12`
SmartWalletFactory V2 (todas las redes)	`0x230b7010529AB6977Dd8581B3eF018ef865BdEf1`
AgentWalletFactory V2 (todas las redes)	`0x8D9e24022777173AD6336e00884b6C87c7EF054c`
SmartWalletFactory V3 (todas las redes)	`0xD42F61AA856A4f47885Ecd2D0ce119411d53C192`
AgentWalletFactory V3 (todas las redes)	`0xd866a7563cDaC9F71423be3332b62c329C676064`
SmartWalletFactory V4 (todas las redes)	`0xfd105197109244483b5f870501326E6faec9F93c`
AgentWalletFactory V4 (todas las redes)	`0xf3A61F9d64d45362E149A111289546523BCd26a6`
ValidatorRegistry (todas las redes, V3 y V4)	`0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2`
HookRegistry (todas las redes, V3 y V4)	`0x551C6e7ABdA04a110790888e711198f25621b066`
Posición LP v4 de prueba (Arbitrum One)	`tokenId 190877 — minteada y quemada por un agente`
Wallet de pruebas V3 (Arbitrum One)	`0x605A4C65dBa64bd26d9F7701e1e4Cda48c594A5c`
Admin de gobernanza (hardware wallet)	`0x3145Bd5e2489d8bDdAb17F23F26F07Ac5aD55A4c`
Redes V2/V3/V4	Arbitrum One · Base · BNB Chain · Ethereum · Polygon · Arbitrum Sepolia
Build V3/V4	solc 0.8.36 · via-IR · optimizer 50 · EVM cancun (congelada para CREATE2)
Herramientas	Slither 0.11.3 · Foundry (forge 1.5.1) · viem 2.x

Arbitrum One mainnet. Las factories V2, V3 y V4 están desplegadas en Arbitrum One, Ethereum, BNB Chain, Base, Polygon y Arbitrum Sepolia, con los validators del Universal Router y del PositionManager v4 registrados en las seis.

G Notas de proceso

Se recoge porque una revisión que solo enumera lo que encontró exagera su propia fiabilidad.

Junto a los hallazgos conviene anotar tres lecciones. La reentrada se descartó una vez, con un razonamiento plausible sobre EOAs, hasta que una prueba de concepto lo contradijo: las suposiciones sobre la forma de una dirección merecen un test, no un argumento. Los problemas más graves de aquí aparecieron al volver a mirar código que ya había pasado una revisión — la apropiación de guardianes vino de una pasada con otra herramienta, la rotación de credencial de un seguimiento de esa misma pasada, y dos más de atacar la recuperación de frente. Y la cifra del análisis estático se movió en uno durante todo el proceso: no subió mientras la reentrada fue explotable durante tres generaciones, ni bajó cuando el arreglo la cerró. Las herramientas encuentran patrones; los hallazgos de aquí salieron de leer el código y preguntarse qué haría un atacante con él. Los contratos siguen siendo de revisión interna y no están auditados externamente.