



BLOCK VENTURE CHAIN CAPITAL

Agent Wallet Security & Test Report

Static analysis, vulnerability remediation & on-chain agent-permission test battery

SUBJECT	BVCC Agent Wallet (ERC-4337 / ERC-7579 smart account)
CODEBASE	Commit a3e28f1 · contracts/src tree 337e84f0 · generation V4
DEPLOYMENT	Arbitrum One · Ethereum · BNB Chain · Base · Polygon · Arbitrum Sepolia
ENGAGEMENT	Internal security review & remediation verification — not an external audit
PERIOD	2026-06-06 → 2026-07-29 · four review rounds
FINDINGS	2 Critical · 3 High · 5 Medium · 1 Low · 2 Informational
STATUS	9 remediated · 4 open or accepted · 303/303 tests

Contents

PART I — AUDIT

1	Executive Summary	Headline results and what a reader should take away
2	Scope, Methodology & Severity Model	What was reviewed, at which commit, how severity is assigned
3	Findings Register	Every finding, its severity and its status, on one page
4	Findings in Detail	BVCC-01 ... BVCC-13
5	Remediation & Deployment Verification	What is live, on which networks, verified how
6	Residual Risk & Recommendations	What is still exposed and what to do about it
7	Conclusion & Limitations	Assessment, and what this review does not establish

PART II — APPENDICES

A	Security Architecture	Agent execution model, call policies, validator registry
B	Test Evidence	Permission battery, signature battery, gas, fork and fuzz
C	Static Analysis	Slither runs and the false-positive register
D	Previously Remediated — V1 & V2	Findings closed in superseded generations, with evidence
E	Remediation Timeline	The four rounds, compressed
F	Key Identifiers & Addresses	Contracts, networks, build
G	Process Notes	What this review got wrong, and what that implies

Findings are cited by identifier (BVCC - 01) throughout, not by section number. Sections are cited by number, appendices by letter.

1 Executive Summary

The BVCC Agent Wallet is an ERC-4337 smart account that lets an owner delegate bounded spending authority to an autonomous agent (an EOA). This report covers four rounds of review of that delegation path, from the first permission battery in June 2026 to the V4 generation deployed on 2026-07-28.

13 FINDINGS, CURRENT CODEBASE	2 CRITICAL	9 REMEDIED	303 TESTS, ALL PASSING
---	----------------------	----------------------	-------------------------------------

What the review found

Two findings are Critical. **BVCC-01**, a cross-function reentrancy, let an agent whose address later acquired code re-enter the owner-level `execute()` and bypass every configured limit at once; a proof of concept moved 49.925 ETH against a 0.003 ETH lifetime budget, and the same attack succeeded against a wallet created from the live V2 factory on a mainnet fork. **BVCC-02**, a wrong command mask in the Universal Router validator, would have accepted a cross-chain bridge deposit as if it were a swap — the exact exfiltration class the validator exists to prevent — and was caught in review before that validator was ever registered on a network.

Three findings are High: agent budgets do not bound protocol spending where a standing token allowance exists (**BVCC-03, open**), guardians could be chosen by whoever first called the permissionless factory (**BVCC-04**), and a completed recovery did not stop the agents authorized before it (**BVCC-05**). The remainder are Medium and below.

What was done about it

Nine findings are remediated and live. Seven of them required a bytecode change and shipped together as the **V4** generation, deployed and independently verified on all six supported networks on 2026-07-28. Deep validation — the layer that inspects calldata a bitmap pin cannot express — is complete: both selectors that defer to the on-chain registry have an active validator on every network, and the full Uniswap v4 liquidity cycle was exercised by an agent on Arbitrum mainnet, ending with the position burned and no native value stranded.

Four findings remain open. One is High and is the material residual risk: where the owner has already granted a token allowance to a protocol, anchoring a call's destination does not bound its value, so a compromised agent key is worth more than its budget. A staged plan is given in **BVCC-03** and the operational mitigation in §6. The other three are Medium or Informational and are accepted, documented limitations.

Two caveats belong next to any statement that V4 fixed these. First, a deployed smart account cannot be upgraded in place: its address derives from its bytecode. **Wallets still on V1, V2 or V3 run the vulnerable bytecode until their owners migrate**, and BVCC-01 affected all three generations alike. Second, BVCC-03 survives V4 and is not a contract matter — zero any standing allowance towards a protocol before authorizing an agent that can reach it.

This is an internal review. It was performed by the development team on its own code, not by an independent third party, and the contracts have not been externally audited. Four of the

findings — BVCC-04, BVCC-06, BVCC-07 and BVCC-11 — came from a later pass over the same code with a different analysis tool, after three earlier passes had missed them. That was still an internal pass: **no independent party has reviewed this code.** §7 states the limits of what this document establishes.

2 Scope, Methodology & Severity Model

Reviewed codebase

ITEM	VALUE
Repository	blockventurechaincapital-crypto/bvcc-wallet-agent
Commit	a3e28f1a7c8e265d5b3630f98e4c8e77c6752eb5
contracts/src tree	337e84f05fa11b948e3ad3221398ea4edc030f45
Build	solc 0.8.36 · via_ir · optimizer_runs = 50 · EVM Cancun (frozen — CREATE2 addresses depend on it)
Tooling	Slither 0.11.3 · Foundry (forge 1.5.1) · viem 2.x

Earlier rounds reviewed the V1, V2 and V3 trees at their own commits; those findings and their remediations are in Appendix D and Appendix E.

In scope

- `BVCCAgentWallet.sol` — agent permission logic (executeAsAgent, per-token & ETH caps, pause, expiry, the call-policy layer).
- `BVCCWallet.sol` — base smart account (execution, fees, guardian recovery).
- `BVCCAgentWalletFactory.sol` / `BVCCWalletFactory.sol` — CREATE2 factories.
- `BVCCValidatorRegistry.sol`, `BVCCUniversalRouterValidator.sol`, `BVCCPositionManagerValidator.sol`, `BVCHookRegistry.sol` and the `IBVCCValidator` interface.
- The call-policy presets the interface registers, as configuration that determines on-chain behaviour.

Out of scope

- The third-party protocols the wallet interacts with (Uniswap, Aave, Permit2) — their own security is assumed, and the wallet is reviewed on the basis that they may behave adversarially at their interface.
- Bundler, paymaster and EntryPoint infrastructure.
- The web interface, except where it registers on-chain policy or is the only place a control exists (noted explicitly where that is the case).
- Economic and market risk, including price manipulation in pools the wallet trades against, except where it is the mechanism of a finding (BVCC-03).

Methodology

- **Manual review** of the agent execution path, permission accounting, recovery and the call-policy layer, in four passes across the four generations.
- **Static analysis** — Slither 0.11.3 over the tree at each round; results and the false-positive register in Appendix C.
- **Unit, fork and fuzz testing** — Foundry. Every finding has a regression test. The router validator is fuzzed over its whole command space against a model of the real router; Aave is exercised on a mainnet fork.

- **Exploit development** — findings are confirmed by a working proof of concept rather than by argument. BVCC-01 was reproduced against a wallet created from the *live* V2 factory on a fork.
- **On-chain testing** — the agent EOA drives `executeAsAgent` directly. Reverts are confirmed by simulation (`eth_call`) against live state; valid cases run as real transactions on Arbitrum Sepolia and Arbitrum One.
- **Deployment verification** — deployed bytecode, owner and registry wiring are read back from each network independently of the deployment script.

Threat model

Stated explicitly because it changes how several findings should be read. A reader who assumes every issue requires stealing a key will misjudge the ones that require nothing at all.

ACTOR	WHAT IS ASSUMED
Owner	Trusted. Holds the passkey. Compromise of the passkey is out of scope — it is equivalent to owning the wallet, and guardian recovery exists for that case.
Agent	Semi-trusted, and assumed hostile for the purpose of this review. Its key lives on a server, in an environment variable or an env file, running unattended; and the agent may belong to a third-party operator the owner does not control. "The agent acts outside its mandate" is not an exotic precondition — it is the event the permission system exists to bound.
Guardian	Honest in the majority. Recovery requires two of three; the design accepts that two colluding guardians take the wallet if the owner does not cancel within 48 hours.
Third party	Untrusted, holds no credential of any kind, and can call every permissionless entry point — including the factories — in any order and on any network.
Protocol	Assumed correct in itself, but may behave adversarially at its interface: a pool may be attacker-controlled, a token may re-enter or return unexpected values.

Why "requires the agent's key" is not a mitigating factor. The per-token caps, daily and lifetime budgets, expiry, pause and recipient whitelist exist precisely because the agent is not fully trusted. If a compromised or hostile agent key were treated as out of scope, the entire permission layer would have no purpose. Findings that hold *only* while the agent behaves are not limits; they are suggestions. This is why BVCC-01 is Critical: it converts "this agent may spend 0.003 ETH" into "this agent may take everything", which is a failure of the product's central promise rather than of one function. The preconditions are recorded per finding in §3 so the reader can weigh each one.

Severity model

Severity is **impact × likelihood**. Impact is the worst outcome for a wallet holder if the issue is exercised; likelihood weighs the preconditions an attacker must satisfy — key compromise, a race, a timelock, a specific wallet configuration — not how probable the attack is thought to be in the abstract.

SEVERITY	MEANING
CRITICAL	Loss of all wallet funds, or full takeover, with preconditions an attacker can realistically reach.
HIGH	Loss of funds beyond the configured limits, or takeover, but gated behind a race, a timelock, an owner action, or a specific configuration.

MEDIUM	Weakens a security control, degrades recovery, or denies service, without directly moving funds.
LOW	Violates a stated invariant with limited practical consequence.
INFORMATIONAL	No impact under current deployment; hardening, or a design decision recorded so it is not silently undone.

A finding's severity is stated as found, not as it stands after remediation. Status is tracked separately so that a fixed Critical is still visible as a Critical that was fixed.

Limitations of this review

Stated here rather than at the end, because it bears on how every finding below should be read. This is an **internal** review: the people who wrote the code also reviewed it. It has not been audited externally. Four of the findings came from a later pass over the V3 code with a different analysis tool, and each had survived three earlier rounds: changing the tool found what repeating the method did not. That pass was internal too — **no independent party has reviewed this code at all**. Absence of a finding in this document is therefore weak evidence of absence. §7 expands on what this review does and does not establish.

3 Findings Register

Thirteen findings in the current codebase, plus two closed in superseded generations (Appendix D). Severity is as found; status is as of 2026-07-29.

ID	FINDING	REQUIRES	SEVERITY	STATUS
BVCC-01	Cross-function reentrancy via <code>_currentAgent</code>	Agent key, or a hostile agent operator	CRITICAL	Fixed in V4
BVCC-02	Universal Router command mask accepts a bridge deposit as a swap	Agent key, with router access	CRITICAL	Fixed pre-deployment
BVCC-03	Token budgets do not bound protocol spending	Agent key + a standing allowance	HIGH	OPEN
BVCC-04	Guardian squatting through the permissionless factory	None — any third party	HIGH	Fixed in V4
BVCC-05	A completed recovery did not stop existing agents	A compromise already under way	HIGH	Fixed in V4
BVCC-06	Unauthenticated <code>credentialId</code> in the factory event	None — any third party	MEDIUM	Fixed in V4
BVCC-07	Credential left stale after a recovery	None — follows any recovery	MEDIUM	Fixed in V4
BVCC-08	Guardians could never be rotated	None — operational	MEDIUM	Fixed in V4
BVCC-09	A single guardian can stall a recovery indefinitely	One malicious guardian	MEDIUM	Accepted
BVCC-10	Pausing or revoking an agent leaves its allowances alive	A spender already approved	MEDIUM	OPEN — interface
BVCC-11	<code>approve</code> could carry ETH past the recipient whitelist	Agent key	LOW	Fixed in V4
BVCC-12	<code>increaseLiquidity</code> has no owner check in the v3 NFPM	Agent key; unreachable as deployed	INFORMATIONAL	Mitigated by exclusion
BVCC-13	Validator constructors accept a zero address	None — deployment error only	INFORMATIONAL	Accepted

By severity and status

SEVERITY	COUNT	STATUS	COUNT
CRITICAL	2	Fixed and deployed	9
HIGH	3	Open	2

MEDIUM	5	Accepted as documented	2
LOW	1		
INFORMATIONAL	2		

Reading the register. Severity is recorded as the issue was found, so a Critical that has been fixed still reads as Critical — that is deliberate, because the reader's question is usually "what was this codebase capable of", not only "what is left". Nine findings are remediated in bytecode that is live on all six networks; the two Critical findings are among them. The one that should drive an owner's behaviour today is **BVCC-03**, which V4 does not close. The **Requires** column matters as much as the severity: six of the thirteen need no credential at all — a stranger who has never held a key can reach BVCC-04 and BVCC-06 — while five presuppose a hostile or stolen agent key, which the threat model in §2 treats as an expected event rather than an excuse.

4 Findings in Detail

Each finding is stated with the severity it had when found, the preconditions an attacker needs, and the evidence that established it. Remediation and the verification of that remediation follow in §5.

The V3 source was put through a further internal pass with a different analysis tool, which raised four issues. All four were reproduced against the code rather than accepted on description; the tests live in `test/ThirdPartyClaims.t.sol`. All four hold, though two are narrower than stated and one is a property of ERC-20 rather than of this wallet. None of them had been found by the three internal rounds.

BVCC-01 **CRITICAL** Fixed in V4

Cross-function reentrancy via `_currentAgent`

LOCATION	BVCCAgentWallet.sol — <code>executeAsAgent</code> / <code>execute</code>
IMPACT	Total loss of wallet funds. Every configured limit is bypassed in a single transaction; a proof of concept moved 49.925 ETH against a 0.003 ETH lifetime budget.
LIKELIHOOD	Medium. The agent address must acquire code after being authorized — an EIP-7702 delegation the agent signs with its own key, or a CREATE2 deployment to a pre-committed address. No owner action is required and nothing warns the owner.
AFFECTS	V1, V2 and V3 alike — confirmed by draining a wallet created from the <i>live</i> V2 factory on an Arbitrum One fork. Wallets that have not migrated to V4 remain exposed.

This result was initially dismissed on the grounds that an agent is an EOA and cannot be handed control mid-transaction. That reasoning does not hold, and the finding is confirmed.

Mechanism

`executeAsAgent` sets `_currentAgent = agent` around its call to `super.execute()`, and `_erc7821AuthorizedExecutor` grants the owner-level `execute()` path to any caller satisfying `caller == _currentAgent`. The parent `execute()` carries no reentrancy guard — `nonReentrant` sits on `executeAsAgent` only — and applies no agent limits, because the limit checks live in `executeAsAgent`. The EOA assumption rests on `AgentMustBeEOA` (`agent.code.length == 0`), which was enforced **only at authorization time** and never re-checked when the agent executed. An address can gain code afterwards: through an EIP-7702 delegation, signed with the agent's own key — the very key the threat model assumes an attacker may hold — or by deploying to a pre-computed CREATE2 address that was code-less when it was authorized.

Scope: V1, V2 and V3 alike. This is not a V3 regression. The four ingredients — the flag, the executor override, the authorization-time-only EOA check and the unguarded parent `execute()` — are present verbatim in all three generations; only the line numbers moved. It was confirmed against deployed code rather than source: on an Arbitrum One fork, a wallet created from the **live V2 factory** (`0x8D9e2402...`) was drained by the same attack (`test/LegacyV2ReentrancyFork.t.sol`).

Reproduction

`test/AgentReentrancyPoC.t.sol`. An agent is authorized while code-less with caps of 0.001 / 0.002 / 0.003 ETH (per-tx / daily / lifetime). Code is then placed at the agent address, modelling the delegation. The agent submits a batch whose only item is a **1 wei** transfer to itself — inside every cap — which hands it control; from there it calls `execute()` back into the wallet and moves

49.925 ETH out of a 100 ETH balance. The 0.075 ETH difference is the wallet's own 0.15% fee, which confirms the drain travelled the owner-level path.

Blast radius

Every agent constraint is bypassed at once — per-transaction, daily, period and lifetime budgets, the recipient whitelist and the V3 call policies. It converts a bounded agent into owner-equivalent control of the wallet, a strictly larger hole than the exfiltration vector V3 was built to close. It does not affect the biometric owner's own path, and it does not let a third party who lacks the agent key do anything.

Remediation — two layers

Three candidate fixes were implemented and measured against three attack shapes rather than argued about. S1 sends 1 wei to the agent directly; S2 leaves the agent code-less on entry and has the batch itself deploy code to that address through a whitelisted helper (no EIP-7702 involved); S3 never names the agent at all, routing value to it through a whitelisted intermediary.

CANDIDATE	S1 DIRECT	S2 CREATE2 MID- BATCH	S3 INDIRECT	FACTORY SIZE
None (pre-fix)	drained	drained	drained	24,283
B — re-check code . length	blocked	drained	blocked	24,294
C — batch may not target the agent	blocked	blocked	drained	24,312
A — guard on external execute ()	blocked	blocked	blocked	24,317
A + B (shipped)	blocked	blocked	blocked	24,328

The two rejected candidates failed for instructive reasons. **B alone** is another point-in-time check — the same class of assumption that caused the bug — and S2 walks straight past it by making the code appear *during* the batch. **C alone** only closes the one door the first proof-of-concept happened to use; S3 reaches the agent through a legitimate protocol instead.

What ships is both layers. **Layer 1** re-checks the EOA invariant on every execution, so it is enforced continuously rather than once. **Layer 2** overrides execute () in the agent wallet to reject any external entry while an agent batch is in flight (ReentrantAgentExecute); executeAsAgent reaches the parent through super . execute () , which resolves statically and therefore skips the override, so the legitimate path is untouched. Layer 2 is the one that does not care *how* or *when* the agent obtained code — it closes the window itself. Each layer has its own regression test, and layer 2 is proven independently through the S2 shape, where layer 1 passes cleanly and the drain is stopped anyway.

Cost: the agent factory grows 45 bytes to 24,328, leaving 248 bytes of EIP-170 headroom, and executeAsAgent case 3 rises from 64,758 to 64,903 gas — 145 gas, 0.2%, because the agent address is already warm as the transaction sender. One deliberate behavioural consequence: an agent that adopts an EIP-7702 delegation for unrelated reasons stops working until it is removed. That is the contract's own stated invariant, now enforced.

Deployed in V4 — but only for wallets created on V4. The fix changes the wallet bytecode and therefore the CREATE2 addresses, so it cannot reach a wallet already deployed; Create2Consistency . t . sol failing against the old constants was the mechanical proof of that. V4 factories are live on all six networks, **and every wallet still on V1, V2 or V3 runs vulnerable bytecode until its owner migrates. The mitigation for those wallets, without redeploying:** authorize agents with a non-empty allowedRecipients that

does not contain the agent's own address, which removes the batch's ability to hand control back to it (`test_RecipientWhitelist_BlocksTheEntry`). Wallets left on the permissive default — an empty recipient list, meaning any destination — are the exposed configuration. Pausing or revoking the agent also closes it immediately.

Among the Low results, 20 `calls-loop` and 4 `timestamp` are inherent to batch execution and to day-indexed limits; `return-bomb` on `_tryBalanceOf` is bounded by the same `PROBE_GAS_CAP` introduced in V2, which also caps how much return data a hostile token can afford to produce. One result is a genuine, if inert, gap:

Accepted nit — no zero-check in the validator constructors. Neither `BVCCUniversalRouterValidator` nor `BVCCPositionManagerValidator` rejects a zero address for the protocol it binds to, unlike the factory constructors hardened in V1 (`ZeroOwner`). A validator bound to the zero address would deny every call — fail-closed, so the deployed behaviour is safe — and the six deployed instances carry the correct constructor arguments, recorded with their codehashes in `contracts/deployments/` and verified on the explorers. Adding the check would change the bytecode, and therefore the `CREATE2` addresses, forcing a redeploy and a fresh 48-hour governance cycle per network. It is recorded here for the next bytecode revision rather than acted on now.

The Informational results are the same classes as in the V1 and V2 runs: inline assembly, low-level calls, naming conventions on immutables, pragma spread across the OpenZeppelin tree, and the `_feeNumerator()` "dead code" report — a known false positive: the function is reached through virtual dispatch and sets the 0.15% agent fee.

Re-run after the V4 fixes: 53 results — 3 High, 27 Low, 23 Informational. The three High results are unchanged, and all three are now false positives: the two uninitialized-state reports are mappings, and `reentrancy-eth` still sees the flag written after the external call even though `BVCC-01` closed the window — it now lists `execute()` among the reachable functions, which is precisely where the guard sits. A second dead-code report appeared for `_afterRecovery()`, the recovery hook of `BVCC-05`: the same false positive as `_feeNumerator()`, an empty virtual overridden in the child. Neither is to be removed. Nothing new was introduced by the fixes. **Static analysis is a pattern detector, not a verdict:** this count did not move while the reentrancy was exploitable for three generations, and it did not move when the fix closed it.

BVCC-02 **CRITICAL** **Fixed pre-deployment**

Universal Router command mask accepts a bridge deposit as a swap

LOCATION	<code>BVCCUniversalRouterValidator.sol</code> — command decoding
IMPACT	Critical. The router would bridge the wallet's funds to an attacker on another chain while the validator classified the call as an ordinary swap — the exfiltration class the validator exists to prevent.
LIKELIHOOD	Medium — a compromised agent key with Universal Router access, and nothing else.
NOTE	Found in review before the validator was registered on any network, so it was never exploitable in production. Recorded because the near-miss is the point: the validator was written to stop exactly this and would have permitted it.

Critical bug caught during review — the command mask. The first version of the validator masked command bytes with `& 0x3f`. The real router masks with `& 0x7f` (bit `0x80` is `FLAG_ALLOW_REVERT`), which means `0x40` is **not** an alias of `0x00` — it is

ACROSS_V4_DEPOSIT_V3, a cross-chain bridge deposit. Under the wrong mask the validator would have accepted it as a v3 swap while the router bridged the wallet's funds to an attacker on another chain: precisely the exfiltration class V3 exists to prevent. The fix replaced masking with exact matching, bound the validator to a single router and rejected empty command strings. The behaviour is now fuzzed across all 256 command bytes against a model of the router, and was re-confirmed on mainnet after deployment (see Appendix B).

BVCC-03 **HIGH** **OPEN**

Token budgets do not bound protocol spending

LOCATION	BVCCAgentWallet.sol — _validateExecutionItem, case 3
IMPACT	Critical where it applies: loss of the entire standing allowance, irrespective of the token budget configured for the agent.
LIKELIHOOD	Medium-low. Requires both a compromised agent key <i>and</i> an allowance the owner has already granted to a protocol the agent can reach. A wallet with no standing allowance is not exposed.
NOTE	The only finding in this report that V4 does not close. See §6 for the mitigation and the staged plan.

The per-token counters accumulate only for the ERC-20 transfer and approve selectors: `_checkTokenWhitelistAndAmount` is reached from those two branches and from nowhere else. A case-3 protocol call therefore spends tokens without touching `tokenMaxAmounts`, the daily limit or the lifetime budget — and without comparing the asset inside its calldata against `allowedTokens`. This is deliberate as far as it goes: the model charges the budget at the approve step, which is what the Aave fork suite asserts, so in the ordinary approve-then-trade flow the budget is consumed.

The gap is **pre-existing allowances**, and the common case is not exotic: an owner who has granted a router or Permit2 a large allowance from the interface — standard practice — leaves the agent able to move that token through case 3 indefinitely without consuming a unit of its token budget. Reproduced in `test_Claim2_PreExistingAllowanceEscapesTokenBudgets`: with all three token caps set to 10, the agent moves 500 through a whitelisted router and both counters stay at zero. Permit2's own `approve(address,address,uint160,uint48)` is likewise a different selector and is not counted.

Severity correction. An earlier draft of this section described the loss as trading beyond the intended volume, on the grounds that call policies pin every recipient to the wallet. That understates it. The pin guarantees *where the output lands*, not *what it is worth*: a compromised agent can swap a valuable asset into a worthless one through a pool it controls, and the worthless token dutifully comes home while the value stays in the attacker's liquidity. Nothing in the current design objects — the Universal Router validator contains no reference to amounts or prices at all, and the SwapRouter02 policies pin the recipient and nothing else, so a swap with `amountOutMinimum = 1` passes every check. Where a standing allowance exists, this is effectively a full drain of the tokens it covers, not slippage.

Token selection narrows this but does not close it. Choosing which tokens an agent may touch does govern transfer and approve, so an unselected token with no standing allowance cannot be moved: the agent cannot approve it either. Two holes remain. A token that already

carries an allowance is reachable regardless of the selection. And Aave's borrow and withdraw consume no allowance at all, so there is nothing to zero: an agent holding an Aave policy can take on debt in an asset the owner never selected, against the owner's collateral, with the liquidation risk that implies. The funds land in the wallet — this is damage rather than theft — but it is damage the token selection was expected to prevent.

Remediation plan

Full in-wallet accounting would require deep validators to decode and return the (token, maxSpend) pairs they see, with the wallet accumulating them through _applyTokenSpend and counting amountInMaximum for exact-output routes. That is a change to every validator plus the wallet, and the agent factory has 101 bytes of EIP-170 headroom. It is therefore staged rather than attempted now.

STAGE	MEASURE	COST
Now interface only	Rename the limit so it stops implying a global cap — it governs direct transfers and approvals, not DeFi volume	None
	At authorization, read the allowances of every relevant token towards every protocol the agent's policies can reach, and require a signed revocation of any that is non-zero	Frontend + one signed batch
	Drive agent operations as atomic batches: approve(exact) → protocol call → approve(0), so the first leg consumes budget and nothing outlives the batch	Frontend; works on today's contract
	For Permit2, check both layers — the ERC-20 allowance to Permit2 and Permit2's internal allowance to the router — and use exact amounts with short expirations	Frontend
	An owner-facing emergency screen that revokes all allowances, and modest balances on wallets that carry agents	Frontend + operational
Next validator layer	Price control in the swap validators: reject a route whose amountOutMinimum is implausible against an oracle or TWAP. A validator is a view function, so it can read one. This is the measure that closes the rigged-pool drain.	New validators; 48 h governance per network
	An Aave validator that checks the asset argument against the wallet's selected tokens, so the selection governs lending too	New validator; same cycle
Later	An agent vault the agent must route through, so the amount handed to it is a physical ceiling and routers leave allowedProtocols entirely	New contract + integration
	In-wallet accounting, once a factory redesign frees the bytecode for it	Requires V5-scale work

How the severity comes down. The rating is driven by two things: the loss has no ceiling, and the precondition is ordinary user behaviour. Note what happens when no standing allowance exists — the agent must approve before it can spend, and that approve is counted against the per-token budget and gated by the token and recipient lists. The budget bounds everything again. The finding is therefore not "the budget does not work" but "the budget does not work *while a standing allowance exists*", and that condition is attackable without touching the contracts.

STATE	WHAT IS BOUNDED	SEVERITY
-------	-----------------	----------

Today	Nothing, where an allowance exists	HIGH
+ allowance hygiene (zero at authorization, exact atomic approvals, short Permit2 expirations)	Cumulative spend, via the token budget	MEDIUM
+ price control in the swap validators	Per-operation loss as well	LOW

Hygiene alone stops at medium because it is a snapshot: an owner can grant an allowance from any dApp afterwards. It lowers the likelihood; only the validator lowers the impact. Two caveats: Aave's `borrow` and `withdraw` consume no allowance, so hygiene does not reach them until an Aave validator checks the asset; and a price bound caps each operation, not the total — bounding the cumulative case-3 spend needs the in-wallet accounting the bytecode cannot currently hold.

What the interface measures cannot do. Blocking unlimited approvals while an agent is active is not enforceable from the wallet's own frontend: an owner can grant an allowance from any dApp, and nothing on-chain prevents it. The authorization-time check is therefore a snapshot that closes the common case — the forgotten standing allowance — not an invariant. Only the validator layer makes it one.

BVCC-04 **HIGH** Fixed in V4

Guardian squatting through the permissionless factory

LOCATION	BVCCWalletFactory.sol / BVCCAgentWalletFactory.sol — <code>createWallet</code> ; BVCCWallet.sol — <code>setGuardians</code>
IMPACT	Critical. Full takeover: the squatter's guardians rotate the owner once the recovery timelock elapses.
LIKELIHOOD	Low. The attacker must deploy the victim's address first on a network the victim has not used, the victim must then fund it, and the 48-hour window must pass without the owner cancelling. Detectable by reading <code>guardians()</code> before funding.
REPORTED BY	Later internal pass, different tooling

A wallet's `CREATE2` address derives from the passkey public key alone; the guardians are not part of it. `createWallet` is permissionless and sets the guardians in the same call, and `setGuardians` is one-shot. Whoever deploys an address first therefore chooses, permanently, the three parties who can rotate its owner. If the wallet already exists the factory returns it unchanged, so a caller arriving second is handed the squatted wallet with no error and no signal.

The full takeover was reproduced end to end

(`test_Claim1_GuardianSquattingTakesOverTheWallet`): an attacker deploys the victim's deterministic address with its own guardians; the victim's app then "creates" the wallet and receives the same address back; two of the attacker's guardians open a recovery, wait out the 48-hour timelock and rotate the signer. The wallet and its balance are theirs.

The multichain design amplifies this precisely because it is deterministic: the passkey public key becomes visible on-chain the moment the user deploys anywhere, and the same address is claimable on every other network. The exposure is therefore **networks where a user has not yet deployed**, not wallets already in use — once correct guardians are set they cannot be replaced. The 48-hour timelock plus `cancelRecovery` is a real defence, but only on a chain the owner is watching, which by definition excludes the chains they have not deployed to.

One part of the reviewer's recommendation does not hold. Rejecting duplicate guardians on-chain is hygiene, not a hole: approvals are tracked per address, so a duplicated guardian cannot approve twice and cannot reach the 2-of-3 threshold alone (`test_Claim1_DuplicateGuardianCannotSelfApprove`). Duplicates degrade the scheme to 2-of-2; they do not enable a solo takeover.

Remediation, applied in source. The factory no longer chooses guardians: it deploys the wallet with none, and `setGuardians` is gated behind `msg.sender == address(this)` — the same self-call pattern `authorizeAgent` and `setCallPolicy` already use — so only a passkey-signed operation can set them. Duplicates are now rejected as well. A squatter is left with an empty shell it cannot configure, and is in effect paying the victim's deployment gas. Guardians are read nowhere except the recovery path, so a wallet without them operates normally; only recovery is unavailable until the owner signs. Seven regression tests cover it, including the owner's real path — a `execute()` batch whose item targets the wallet itself — and the fact that an authorized agent cannot reach it either, because `executeAsAgent` refuses any item targeting the wallet.

BVCC-05 **HIGH** **Fixed in V4**

A completed recovery did not stop existing agents

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

IMPACT High. The attacker's agent keeps spending after the owner believes the attacker has been locked out — and compromise is precisely why an owner reaches for recovery.

LIKELIHOOD Medium — applies to any recovery performed in response to a compromise, which is the main reason the mechanism exists.

Recovery did not revoke agents. **FIXED IN V4** `executeRecovery` rotated the signer and touched nothing else, so an agent authorized before the recovery kept spending after it, budget intact. That matters because compromise is precisely why an owner reaches for recovery: an owner who recovers believing they have locked the attacker out had not revoked the attacker's agent. Recovery now pauses agents through an `_afterRecovery` hook. Pausing rather than revoking is deliberate — constant gas whatever the agent list looks like, reversible, and it preserves each agent's configuration for the new owner to judge rather than forcing them to rebuild it. The guard is `if (!paused()) _pause()`: without it, a wallet whose agents were already paused could not have completed a recovery at all.

BVCC-06 **MEDIUM** **Fixed in V4**

Unauthenticated `credentialId` in the factory event

LOCATION Factory walletCreated event; BVCCWallet.sol — `_setCredential / setCredentialId`

IMPACT Medium. A squatter could publish a bogus credential for someone else's address, and after a recovery the on-chain credential still pointed at the passkey that had just been replaced — the interface would offer the wrong key.

LIKELIHOOD Medium — the first follows from BVCC-04; the second from any completed recovery.

REPORTED BY Later internal pass, different tooling — follow-up review

Two related findings surfaced while fixing it. The `credentialId` travelled through the factory as an unauthenticated event field, so a squatter could publish a bogus one for someone else's address. It cannot cost funds — signatures are verified against the stored public key — but a

client that trusts it pre-selects a credential the owner does not hold, and the passkey prompt fails. It has been moved out of the factory entirely: the wallet now announces it itself, in the same passkey-signed call that sets the guardians, through `CredentialSet(bytes32 indexed credentialHash, bytes credentialId)`. The hash is indexed so a client holding a passkey `rawId` can find the wallet it belongs to.

BVCC-07 **MEDIUM** **Fixed in V4**

Credential left stale after a recovery

LOCATION	BVCCWallet.sol — <code>executeRecovery / setCredentialId</code>
IMPACT	Medium. After a guardian recovery the on-chain credential still pointed at the passkey the owner no longer held, so a client trusting it prompts for the wrong key and the recovered wallet appears unusable.
LIKELIHOOD	Certain after any completed recovery — not an attack, a broken end state for the mechanism that exists to rescue the wallet.
REPORTED BY	Later internal pass, different tooling — follow-up review

Discovered while fixing BVCC-04. `executeRecovery` rotates the signer but left the credential untouched, so after a guardian recovery the on-chain credential pointed at a passkey the owner no longer held; `setCredentialId`, self-call gated, closes that.

BVCC-08 **MEDIUM** **Fixed in V4**

Guardians could never be rotated

LOCATION	BVCCWallet.sol — <code>setGuardians</code>
IMPACT	Medium. A guardian whose key was later lost or compromised was permanent for the life of the wallet; the only remedy was to create a new wallet and move the funds.
LIKELIHOOD	Low as an attack, certain as an operational failure over a long enough horizon.

Guardians could never be rotated. **FIXED IN V4** `setGuardians` was one-shot, so a guardian whose key was later lost or compromised was permanent for the life of the wallet and the only remedy was to create a new wallet and move the funds. The set is now replaceable by the owner. It remains a self-call, so the squatting of BVCC-04 stays closed, and rotation is refused while a recovery is in flight (`RecoveryActive`) — otherwise a stolen passkey could swap the guardians out from under a recovery already under way. The owner cancels first, then rotates.

BVCC-09 **MEDIUM** **Accepted**

A single guardian can stall a recovery indefinitely

LOCATION	BVCCWallet.sol — <code>initiateRecovery</code>
IMPACT	Medium. Denial of service against the rescue mechanism. No funds move and the owner is never rotated.
LIKELIHOOD	Medium — requires one malicious or compromised guardian willing to front-run each second approval.
NOTE	Accepted as a documented limitation: it is a race rather than a lock, and once the honest pair lands two approvals nobody can reset it.

A single guardian can stall a recovery indefinitely. **OPEN — ACCEPTED** `initiateRecovery` may be called whenever approvals are below two, and it overwrites the pending key and zeroes

the timelock. One malicious guardian therefore only has to front-run each second approval to keep the honest pair from ever reaching the threshold. This is denial of service against the rescue mechanism, not theft: the owner never rotates. It is also a race rather than a lock — a companion test shows that once the honest pair lands two approvals, no one can reset it. Accepted as a documented limitation.

The trust model, stated plainly. Two colluding guardians take the wallet if the owner does not cancel inside the 48-hour window — that is what makes recovery work, and it is tested as such. It means the owner must be watching every network the wallet exists on, which is the same operational requirement the guardian squatting of BVCC-04 exposed from a different direction.

BVCC-10 MEDIUM OPEN – interface

Pausing or revoking an agent leaves its allowances alive

LOCATION	ERC-20 allowance semantics; interface revocation flow
IMPACT	Medium. An owner who pauses a compromised agent has not stopped the spender it already approved. The allowance survives in the token contract.
LIKELIHOOD	Medium — every revocation performed under the reasonable assumption that it is enough.
NOTE	Inherent to ERC-20: nothing the wallet does can retract an allowance held by the token. The remedy is an interface that revokes explicitly — see §6.

An allowance lives in the token contract, so nothing the wallet does can retract it. Reproduced in `test_Claim3_AllowanceSurvivesPauseAndRevoke`: the agent approves within its cap, the owner pauses *and* revokes, and the spender still pulls the full amount. The reviewer correctly notes the amount was charged to the budget at approval time, so no limit is evaded. This is a property of every smart account, not a defect of this one, and it matters for incident response rather than for the permission model: after revoking an agent, an owner should also zero the allowances it was granted. An explicit warning at pause time and an owner-facing allowance revocation screen cover it.

BVCC-11 LOW Fixed in V4

approve could carry ETH past the recipient whitelist

LOCATION	BVCCAgentWallet.sol — <code>_validateExecutionItem</code>
IMPACT	Low. Violates the contract's own stated invariant that token calls carry no value; the practical consequence is bounded by the ETH budget, which still applies.
LIKELIHOOD	Low — requires a compromised agent key and a token that accepts value-bearing calls.
REPORTED BY	Later internal pass, different tooling

An execution carrying both a positive `value` and `approve` calldata is classified by the agent validator as a token operation — so the recipient check is applied to the *spender* — while the parent `execute()` falls through to case 3 and forwards the ETH to the token address. The ETH caps still apply, since `value` is accumulated unconditionally; what is skipped is the destination whitelist. Reproduced in `test_Claim4_ApproveCarriesEthPastRecipientWhitelist`, where a plain ETH send to the same address reverts `RecipientNotAllowed` and the same ETH riding on an `approve` arrives.

Severity is limited by a detail the reviewer does not mention: the target's approve must be payable for the call to succeed at all. Standard ERC-20s are not, so the attempt simply reverts; the test needs a purpose-built payable token to demonstrate the path. Exploitation therefore requires the owner to have whitelisted an unusual or hostile token. A related piece of untidiness sits next to it: a transfer carrying value is executed by the parent without forwarding the ETH, yet the agent still charges that value to its ETH budget. Both are now refused: a token call carrying value reverts `TokenCallWithValue`, verified by a regression test that also confirms an ordinary value-free `approve` still works.

BVCC-12 INFORMATIONAL Mitigated by exclusion

increaseLiquidity has no owner check in the v3 position manager

LOCATION	Uniswap v3 NonfungiblePositionManager; call-policy presets
IMPACT	High if it were whitelisted: unlike <code>decreaseLiquidity</code> , <code>collect</code> and <code>burn</code> , the position manager performs no owner check on <code>increaseLiquidity</code> . An agent could pull the wallet's approved tokens into a position owned by the attacker — a recipient-whitelist bypass that cannot be pinned, because <code>word 0</code> is the <code>tokenId</code> , not an address.
LIKELIHOOD	Not reachable as deployed — the selector is deliberately absent from every policy preset.
NOTE	Recorded so the exclusion is not silently undone by someone adding the selector for convenience. To add liquidity, the agent mints a fresh position with the recipient pinned to the wallet. Re-enabling it needs a deep validator that checks <code>ownerOf(tokenId) == wallet</code> .

BVCC-13 INFORMATIONAL Accepted

Validator constructors accept a zero address

LOCATION	<code>BVCCUniversalRouterValidator.sol</code> , <code>BVCCPositionManagerValidator.sol</code> — constructors
IMPACT	None as deployed. A validator constructed with a zero router or position manager would match nothing and deny every call — a failure in the safe direction.
LIKELIHOOD	Deployment-time error only; not attacker-controlled.
NOTE	Every deployed instance was read back on all six networks and both immutables are non-zero. Accepted rather than fixed: adding the check would require redeploying and re-registering validators that are demonstrably correct.

5 Remediation & Deployment Verification

Nine findings are remediated. Seven required a bytecode change and shipped together as the V4 generation; the other two were closed before the affected code reached a network. This section records what is live, where, and how that was established independently of the deployment script.

Seven findings required a bytecode change. All seven are fixed and **live on all six networks**. Because CREATE2 addresses derive from the bytecode, they could not be applied to wallets already deployed: the migration is the same playbook as V1 → V2 and V2 → V3 — users recreate their wallet and move funds, after which the superseded factories are killed. They travelled in one deployment rather than forcing several migrations. The contracts were renamed to V4 and the EIP-712 domain follows (BVCCSmartWalletV4), as in the V2 → V3 generation; the interface reads that domain to detect an outdated wallet and offer the migration.

ITEM	STATUS
Cross-function reentrancy (BVCC-01)	Fixed two layers, each with its own regression test
Guardian squatting (BVCC-04)	Fixed the factory no longer chooses guardians; setGuardians is passkey-gated
Unauthenticated credentialId (BVCC-06)	Fixed moved out of the factory into a wallet-emitted, signed event
Credential stale after recovery (BVCC-07)	Fixed setCredentialId, self-call gated
approve carrying ETH (BVCC-11)	Fixed token calls must carry zero value
Agents survive recovery (BVCC-05)	Fixed recovery pauses agents
Guardians could not be rotated (BVCC-08)	Fixed owner may replace the set, never during a recovery
Case-3 token accounting (BVCC-03)	Open staged; the validator stage is what closes it
Single-guardian recovery stall (BVCC-09)	Open accepted; denial of service, not theft
Zero-check in validator constructors (BVCC-13)	Open accepted; inert — every deployed instance was read back and both immutables are non-zero
Allowance revocation UX, token-cap defaults (BVCC-10)	Open interface work — deliberately not put in the contract

V4 deployment — Arbitrum One · Ethereum · BNB Chain · Base · Polygon · Arbitrum Sepolia

CONTRACT	ADDRESS (CREATE2, IDENTICAL ON EVERY NETWORK)
SmartWalletFactory V4 (salt ...0A)	0xfd105197109244483b5f870501326E6faec9F93c
AgentWalletFactory V4 (salt ...0B)	0xf3A61F9d64d45362E149A111289546523BCd26a6
ValidatorRegistry	0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2 — unchanged

Verified independently of the deployment script by reading each network back: the deployed bytecode is identical across all six (smart wallet 15,959 bytes, agent wallet 24,541), the owner is the governance hardware wallet 0x3145Bd5e...A4c, and the registry is untouched. The factories are verified on the block explorers.

The registry is deliberately untouched: it is the address compiled into every wallet as a constant, and the freeze-guard test proves the V4 wallet still resolves to it. Validators are shared across generations, so nothing had to be re-registered for V4.

Liquidity on v4 — exercised end-to-end by an agent on mainnet

The deep-validation path that V3 introduced and V4 inherits was run in full by an agent on Arbitrum One, against a native pool — the case bitmap pinning cannot express, and the reason BVCCPositionManagerValidator exists. Pool ETH/USDC, fee 500, tick spacing 10, no hooks, full range.

STEP	TRANSACTION	GAS	RESULT
mint	0xb4245927...f1269	622,967	tokenId 190877, ownerOf = wallet
read position	—	—	currency0 native, currency1 USDC, liquidity 4,377,897,532
collect	0x9296f6f3...a865c7	218,749	success, zero proceeds, zero fee
decrease 100% + burn	0x92cce43d...dca9ef	267,739	NFT burned

The validator behaved as designed throughout: the TAKE_PAIR and SWEEP actions were accepted with recipient resolving to the wallet, and the dry-run passed as soon as the PositionManager was added to allowedProtocols. The mint sent 0.0001 ETH and the SWEEP returned the 17,050 wei of native surplus, so the PositionManager was left holding **zero ETH** — the stranded-value case the validator refuses when no native SETTLE consumes the value. Final state re-read on-chain for this report: ownerOf(190877) reverts NOT_MINTED, position liquidity is 0, PositionManager native balance is 0.

This cycle was exercised on **Arbitrum One only**. On the other five networks the validators are deployed, registered and confirmed bound to the correct PositionManager, but the end-to-end liquidity cycle has not been run. The claim this report makes is therefore: proven end-to-end on Arbitrum, deployed and verified identical elsewhere.

The agent factory ends at **24,541 bytes**, 35 below the EIP-170 limit, with executeAsAgent case 3 at 64,972 gas. That margin is thin, and it is the reason two candidate hardening measures were deliberately kept out of the contract: requiring non-zero token caps when an agent has protocol access, and capping the owner's own approvals while an agent is active. Both belong in the interface, where they cost nothing and can explain themselves. The remaining bytecode is reserved for measures with no off-chain equivalent.

6 Residual Risk & Recommendations

Four findings are not closed by V4. One of them is the material risk to a wallet holder today; the other three are accepted limitations or interface work. This section states what an owner should actually do.

The one that matters: BVCC-03

V3 anchored the destination of an agent's protocol calls, and that holds. But anchoring a destination is not the same as bounding a value. Where the owner has already granted a token allowance to a protocol the agent can reach, a compromised agent key can swap a valuable asset for a worthless one in a pool the attacker controls: the destination anchor sends the proceeds home, and the value stays with the attacker. No validator inspects amounts or prices.

What an owner should do, today, with no contract change: zero any standing allowance towards the protocols an agent can reach *before* authorizing it, and prefer per-transaction approvals over standing ones. A wallet with no standing allowance is not exposed to this.

The staged plan. The interface can narrow it now — mandatory non-zero token caps when an agent has protocol access, revocation of existing allowances at authorization time, an emergency revocation screen. What closes it properly is the validator layer: a deep validator with an oracle or TWAP reference that rejects a swap whose output is implausible against the market price. A vault or internal accounting model would close it completely and is the longest-horizon option. The first stage is interface work and is deliberately **not** put in the contract — see the bytecode budget in §5.

Accepted limitations

- **BVCC-09** — a single guardian can stall a recovery. Denial of service against the rescue path, not theft. Choose guardians accordingly: the mechanism assumes at least two of the three are honest and reachable.
- **BVCC-10** — pausing an agent does not retract its allowances. Revoke allowances explicitly; pausing alone is not containment.
- **BVCC-13** — zero-address checks absent from validator constructors. Inert as deployed, verified on all six networks.

Operational recommendations

1. **Migrate.** A deployed wallet cannot be upgraded in place. Funds left on V1, V2 or V3 run bytecode with BVCC-01 unfixed.
2. **Authorize agents with a non-empty recipient whitelist** that excludes the agent's own address. On unmigrated wallets this is what closes BVCC-01; on V4 it remains good practice.
3. **Zero standing allowances** before authorizing an agent — the BVCC-03 mitigation, and the only one on this list that V4 does not make redundant.
4. **Verify a wallet's guardians on each network before funding it there.** Redundant on V4, still necessary for wallets created by an older factory.
5. **Deploy proactively on every supported network** so no address is left unclaimed.

V4 removes the need for three of the four interim mitigations this report previously carried: guardian squatting is closed at the factory, the credential is wallet-emitted, and a recovery now pauses agents. One survives V4 and is not a contract matter — **zero any standing allowances towards the**

protocols an agent can reach before authorizing it. That is the BVCC-03 case-3 exposure, and it stays open until the validator stage lands.

Users on V1, V2 or V3 remain on the old bytecode until they migrate. A deployed wallet cannot be upgraded in place, so every finding above still applies to funds left behind — including the cross-function reentrancy of BVCC-01, which was exploitable on all three earlier generations. The interface detects an outdated wallet from its EIP-712 domain and prompts the migration; the superseded factories are killed only once users have moved.

7 Conclusion & Limitations

The BVCC Agent Wallet enforces a comprehensive, layered permission model. Three high-severity issues were identified across three review rounds, each reproduced, remediated, redeployed and re-verified, and the on-chain permission battery passes on the patched bytecode. Four more surfaced while preparing this edition — a cross-function reentrancy, guardian squatting through the factory, the unbounded case-3 spending of BVCC-03, and agents surviving a recovery. All but one are fixed and deployed in the V4 factory generation, together with the credential, guardian-rotation and approve-with-value fixes. Guardian recovery was executed and verified end-to-end — ownership rotation to a new passkey plus on-chain rejection of the old key (see Appendix D). The V2 round fixed a mainnet gas-griefing issue in the fee snapshot and concluded with a deterministic multichain deployment (see Appendix D).

The V3 round is the deepest change to the agent path so far. It closed a fund-exfiltration vector that budgets could not see, by making agent DeFi calls default-deny per selector and anchoring the destination either to the wallet itself or to an on-chain validator whose registration is asymmetric — immediate to deny, 48 hours to allow (see Appendix A, Appendix A). Two would-be bypasses were caught inside that work: a command mask that would have accepted a cross-chain bridge deposit as a swap, and an `increaseLiquidity` path with no owner check, which was deliberately left unregistered. The model is verified by 264 unit, fork and fuzz tests, by a mainnet swap in which every recipient resolves to the wallet and the fee is exact, and by the adversarial variant of that same swap being rejected before execution (see Appendix B).

V4 closed that round and is live on all six networks, with the deep-validation layer complete: both selectors that defer to the registry — the Universal Router's `execute` and the v4 PositionManager's `modifyLiquidities` — have an active validator everywhere, and the full liquidity cycle was exercised on Arbitrum mainnet by an agent, ending with the position burned and no native value stranded in the PositionManager (see Appendix A, Appendix D).

The residual risk is stated plainly rather than minimised. Case-3 protocol calls are safe in the sense V3 set out to make them — the destination is anchored — but anchoring the destination is not the same as bounding the value, and where a standing allowance exists that difference is the whole loss. That gap survives V4: a compromised agent key on a wallet left in the permissive default, with an allowance already granted, is still worth more than its budget. The staged plan in BVCC-03 says which part the interface can narrow now and which needs the validator layer to close properly. Separately, every finding in this report still applies to funds sitting in V1, V2 or V3 wallets, because a deployed wallet cannot be upgraded in place — the exposure ends when the user migrates, not when V4 ships.

What this review does and does not establish

It establishes that the specific findings recorded here were reproduced, remediated where stated, and that the remediation is present in bytecode deployed on the networks listed in Appendix F — each read back from chain rather than inferred from a deployment script. Every finding has a regression test, and the suite passes.

It does **not** establish that the contracts are free of other defects. This is an internal review by the team that wrote the code. The four findings contributed by a later pass with different tooling — BVCC-04, BVCC-06, BVCC-07 and BVCC-11 — had each survived three earlier rounds, which is the clearest available measure of what repeating the same method misses. **No independent party has reviewed this code**, so nothing here has been checked by someone with an interest in finding fault.

The contracts have not been externally audited, and this document should not be read as a substitute for one.

Coverage is also uneven by network. The permission battery ran on Arbitrum Sepolia; the call-policy layer and the Uniswap v4 liquidity cycle were exercised on Arbitrum One. On the other four networks the contracts are deployed and verified byte-identical, and the validator wiring was read back, but no end-to-end cycle was run.

A Security Architecture

Reference material for the mechanisms the findings refer to: how an agent call is validated, what a call policy encodes, and how the on-chain validator registry works.

Agent execution model

The agent calls `executeAsAgent(bytes32 mode, bytes executionData)` as an ordinary EOA transaction (not a `UserOp`). `mode` is the ERC-7579 batch selector (`0x0100...00`); `executionData` is an ABI-encoded `Execution[]` of (`target`, `value`, `callData`). Validation order inside the call is **per-tx** → **recipient/protocol** → **period** → **daily** → **total**, with state updated **before** external execution (checks-effects-interactions) and a `nonReentrant` guard.

HIGH (agent fund exfiltration)

FIXED — V3 DEPLOYED 2026-07

Description

Through V2 an agent's budget was charged whenever value left the wallet along a path the wallet could see: a native transfer, an ERC-20 transfer, or an approve. A protocol call (case 3) was permitted as soon as its target appeared in `allowedProtocols`. That is not enough, because several DeFi entry points take the destination as an argument:

- Aave's `Pool.withdraw(asset, amount, to)` sends the supplied position straight to `to` — no approve, no transfer, so the budget never saw it.
- Swap and LP calls (`exactInputSingle`, Universal Router `execute`, `mint`) carry a recipient argument the agent chose freely.

A stolen or misbehaving agent key could therefore name its own address as the destination and drain everything the wallet had supplied or approved to a whitelisted protocol, **without consuming a single wei of its budget**. Severity is HIGH: the loss is bounded only by the wallet's DeFi exposure, and the on-chain accounting shows nothing.

Remediation (V3) — per-selector call policies

Case-3 calls are now **default-deny per selector**. Whitelisting the protocol is no longer sufficient: the owner must also register a policy that says *where the money is allowed to go*. Policies are written with `setCallPolicy(target, selector, policy)`, callable only by the wallet itself — in practice the owner's passkey, bundled into the same signature that authorizes the agent. The policy is one `uint256` word:

BITS	FIELD	MEANING
255	POLICY_ALLOWED	Selector is permitted. Absent → <code>SelectorNotAllowed</code>
254	POLICY_DEEP	Route the call to the on-chain validator registry (see Appendix A)
223..192	PIN_WALLET (32-bit bitmap)	Calldata word <i>i</i> must equal <code>address(this)</code>
191..160	PIN_PROTOCOL (32-bit bitmap)	Calldata word <i>i</i> must be an address in <code>allowedProtocols</code>

Enforcement inside `executeAsAgent` runs before any external call: (1) the selector is read from the `calldata` (< 4 bytes → `calldataTooShort`); (2) the policy must carry `POLICY_ALLOWED`; (3) every pinned word is compared **full-width**, which also rejects dirty high bits, and a word read past the end of the `calldata` yields zero and therefore fails — the check is fail-closed by construction; (4) if `POLICY_DEEP` is

set, the registry must return true. Four custom errors were added: SelectorNotAllowed, PinnedArgMismatch, CalldataTooShort and PolicyValidationFailed. One related default also changed: an empty allowedProtocols list now reverts NoProtocolsWhitelisted instead of meaning "any protocol".

The owner is never subject to call policies. They apply only to agents. A human signing with their passkey keeps using any dApp normally — V3 narrows the delegated path, not the owner's.

Registered policies (Arbitrum One shown; the same shapes are used on the other five networks)

PROTOCOL	SELECTORS	ANCHOR
Uniswap SwapRouter02	exactInputSingle · exactOutputSingle	PIN_WALLET word 3
Uniswap SwapRouter02	exactInput · exactOutput (multi-hop)	PIN_WALLET word 2
Aave v3 Pool	supply · withdraw · borrow · repay	PIN_WALLET word 2 / 2 / 4 / 3
Aave v3 Pool	repayWithATokens · setUserUseReserveAsCollateral · setUserEMode	Allowed, no pin — no destination argument exists
Permit2	approve(address,address,uint160,uint48)	PIN_PROTOCOL word 1 (the spender)
Uniswap Universal Router	execute(bytes,bytes[],uint256)	DEEP → validator registry
Uniswap v3 NFPM	mint · collect	PIN_WALLET word 9 / 1
Uniswap v3 NFPM	decreaseLiquidity · burn	Allowed, no pin — the NFPM gates these on tokenId ownership
WETH	deposit · withdraw	Allowed, no pin — both credit/pay msg.sender

Two calldata traps found while writing the presets.

- **Offset slide.** The multi-hop exactInput struct carries a bytes path, which makes the tuple dynamic and slides an offset word in front of it: the recipient sits at word 2, not word 3 as in the *Single variants. Pinning word 3 there would have pinned amountIn and left the destination free. Every offset was re-derived by encoding real calldata.
- **increaseLiquidity is deliberately not whitelisted.** Unlike decreaseLiquidity, collect and burn, the v3 NFPM applies no owner check to it — only a deadline check. An agent could therefore have pushed the wallet's NFPM-approved tokens into an attacker-owned tokenId, a recipient bypass that no bitmap pin can express because word 0 is a tokenId, not an address. Agents add liquidity by minting a fresh position with the recipient pinned to the wallet; re-enabling increaseLiquidity would require a deep validator that checks ownerOf(tokenId) == wallet.

What each layer covers

CASE	OPERATION	GOVERNED BY
1	Native ETH transfer	allowedRecipients + native limits
2	transfer(to, amount)	allowedRecipients + token limits
2b	approve(spender, amount)	allowedRecipients + token limits
3	DeFi / protocol call	allowedProtocols + call policies

Scope note. V3 closes fund exfiltration through swaps, LP and protocol withdrawals: case 3 is now safe by construction, whatever the owner configures. It does not turn a compromised agent into a no-op — a direct transfer or approve (case 2/2b) remains governed by the recipient whitelist and the per-token limits, and those are the owner's choice. Simulated on mainnet against the test agent, with an empty recipient list and limits left at zero, a direct USDC.transfer to an attacker passes; with recipients and limits configured it reverts RecipientNotAllowed. The on-chain limits are the security model.

Why a second mechanism

A bitmap pin only reaches a word at a fixed offset. In the Universal Router's execute(bytes commands, bytes[] inputs, uint256 deadline) — and in Uniswap v4's modifyLiquidities — the recipient is buried inside variable-length data, so no fixed offset exists. Those selectors carry the DEEP bit instead, which defers the decision to a stateless on-chain validator.

Design

- **Fixed dispatch address.** BVCCValidatorRegistry is compiled into the wallet as a constant (0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2). Because the policy word carries only a flag and never an address, an owner cannot be tricked into pointing a policy at an attacker's validator. A dedicated test (Create2Consistency.t.sol) fails the build if that constant ever drifts from the registry's CREATE2 address.
- **Fail-closed dispatch.** No validator registered for the target → false. A validator that reverts, or that has no code, denies as well. Only an explicit true lets the call through; anything else reverts PolicyValidationFailed.
- **Stateless validators.** Validators are view contracts reached by staticcall. They never hold funds, never receive approvals and cannot mutate state — they answer yes or no.
- **Dual consent.** Enabling a complex protocol takes two independent steps: BVCC registers the validator in the registry, and the owner registers the ALLOWED|DEEP policy with their passkey. Either one missing means the call is denied.

Asymmetric governance

DIRECTION	FUNCTION	DELAY
Allow — register or replace a validator	proposeValidator → activateValidator	48 h timelock + ValidatorProposed event
Deny — cancel a pending proposal	cancelProposal	Immediate
Deny — disable an active validator	freezeValidator	Immediate

The delay runs only in the permissive direction, so a malicious or mistaken proposal is public for two days before it can take effect and owners can react (pause or revoke their agents). The worst case

with a stolen admin key is therefore denial of service, or — after 48 publicly visible hours — a degradation to V2-level protection on one protocol. It is never a direct path to funds: the registry moves nothing.

The Universal Router validator

- **Bound to one router.** The router address is an immutable constructor argument; a call with any other target returns `false`. A new router needs its own validator and its own 48-hour governance cycle.
- **Exact command-byte matching** over `{0x00 V3_SWAP_EXACT_IN, 0x01 V3_SWAP_EXACT_OUT, 0x10 V4_SWAP}`. Every other byte is denied, including allow-revert variants and unknown commands.
- **Every recipient is checked**, including the v4 TAKE action (`{0x07 SWAP, 0x0b SETTLE, 0x0e TAKE}`): it must be the wallet or the `MSG_SENDER` sentinel. `SWEEP`, `TRANSFER`, `PAY_PORTION`, `WRAP` and `UNWRAP` are denied outright.
- **Malformed input denies.** Undecodable calldata reverts inside the validator, which the registry treats as a denial; an empty `commands` string is rejected explicitly.
- **Scope.** Token → token swaps. Native legs relying on the router's `ADDRESS_THIS` are deliberately out of scope; native swaps are executed as BVCC's own batches through `WETH`, so the router never holds wallet funds mid-flight.

Critical bug caught during review — the command mask. The first version of the validator masked command bytes with `& 0x3f`. The real router masks with `& 0x7f` (bit `0x80` is `FLAG_ALLOW_REVERT`), which means `0x40` is **not** an alias of `0x00` — it is `ACROSS_V4_DEPOSIT_V3`, a cross-chain bridge deposit. Under the wrong mask the validator would have accepted it as a v3 swap while the router bridged the wallet's funds to an attacker on another chain: precisely the exfiltration class V3 exists to prevent. The fix replaced masking with exact matching, bound the validator to a single router and rejected empty command strings. The behaviour is now fuzzed across all 256 command bytes against a model of the router, and was re-confirmed on mainnet after deployment (see Appendix B).

Aave needs no validator

Aave's destination arguments (`onBehalfOf`, `to`) sit at fixed offsets, so they are handled by bitmap pins inside the wallet itself. No Aave policy carries the `DEEP` bit, the registry is never consulted for Aave, and no timelock is involved. Only operations whose safety cannot be phrased as "word *N* must be the wallet" — flash loans, multicall, adapter routes — would require a validator, and those remain out of scope.

Registered validators — complete on every network

Exactly two selectors carry the `DEEP` bit and therefore consult the registry:
`execute(bytes,bytes[],uint256)` on the Universal Router, and `modifyLiquidities(bytes,uint256)` on the Uniswap v4 PositionManager. Both are registered and active on all six networks, read back from `validators(address)` on the registry:

NETWORK	UNIVERSALROUTER VALIDATOR	POSITIONMANAGER V4 VALIDATOR
Arbitrum One	0xD1522594e185C882bfDEc6FFD4DE8a53F69AF0Ca	0x6aE9c36121D5cD2d4Bde816F7F857AfB36Ccb8Eb
Ethereum	0x3615a0Fc0663C0201B543deC9816d5Ef30a82ffb	0xeFd47Aeb41e532D6a4ed04553b25827d88a4364d
BNB Chain	0x6945f861e0D7FCD566739000f3fDC1D44896D815	0x5f5Ee1068c2c54bBE896A08444958a289407Cd35

Polygon	0xC1C584E4149eD2EFfe20DA0155b1B8257232102fF	0x62Eb47697BcF63dC9450B662d259AFf555958e25
Base	0x4A2Fc73AADEfC84513defaa0c444d2150AC6A6D8	0xa2F35323DAcBFC2b5909B15607b511c6953e78a5
Arbitrum Sepolia	0x87550034627966e32d82E3b8AdB3f19c62E8d86E	0x32aA1B3E35dc6fF09EB7CDc8F6b43F3573AD2182

Each PositionManager validator was checked individually against the v4 PositionManager it is bound to through its POSITION_MANAGER immutable, and every one resolves to the correct chain-specific address. All six share the same H00K_REGISTRY (0x551C6e7ABdA04a110790888e711198f25621b066, CREATE2, code present on each). A crossed binding would have failed closed and denied all liquidity operations silently, which is why it is verified per network rather than inferred from the deployment script.

The Uniswap **v3** NonfungiblePositionManager has **no** registered validator on any network, and that is correct rather than a gap. Its policies are bitmap pins — `mint` anchors the recipient at word 9, `collect` at word 1, `decreaseLiquidity` and `burn` are owner-gated by the NFPM itself — so no v3 policy carries the `DEEP` bit and the registry is never consulted.

BVCCPositionManagerValidator is a v4 contract: it is bound to a single v4 PositionManager and decodes v4 action bytes. Registering it against the v3 NFPM would break v3 liquidity, not secure it. `increaseLiquidity` remains deliberately unregistered for the reason given in Appendix A — the NFPM performs no owner check on it.

B Test Evidence

The test batteries behind the findings and their remediations. Every finding in §4 has a regression test; these are the batteries that exercise the model as a whole.

Executed against the patched wallet `0x1ED261...F0C3` with the agent EOA `0x3852...4aB4`. Reverts confirmed by simulation; valid cases as real transactions. Allowed recipients were the two recovery guardians (bot-controlled, funds recoverable).

#	TEST	EXPECTED	RESULT
1	ETH per-tx ($0.0002 > \max 0.0001$)	revert	ExceedsPerTxLimit
2	ETH within limit (0.0001)	pass	Success
3	USDC per-tx ($2 > \max 1$)	revert	ExceedsTokenMaxAmount
4	USDC daily ($1+1+1$, cap 2) — real txs	3rd reverts	TokenDailyLimitExceeded
5	Token send to non-allow-listed dest	revert	RecipientNotAllowed
6	ETH send to non-allow-listed dest	revert	RecipientNotAllowed
7	Total budget ($0.0002+0.0001 = 0.0003$) — real txs	next reverts	AgentBudgetExceeded
8	Period budget ($0.0003+0.0002 = 0.0005$) — real txs	next reverts	PeriodBudgetExceeded
9	Period auto-rollover (periodStart 0)	window resets on 1st spend	Verified
10	Paused (owner pauses)	agent blocked	EnforcedPause
11	Expiry (lapsed naturally)	agent blocked	AgentPermissionsExpired

Observation. The recipient allow-list applies to **token transfers** as well as ETH — a token send to a non-allow-listed address reverts `RecipientNotAllowed`. Re-authorization via “Edit” does not reset the daily counter (keyed by UTC day); with `periodStart = 0` the first spend rolls the period window.

Privilege-escalation resistance (from the 06-06 session)

An agent targeting the wallet itself to call owner functions (`pauseAgents`, `revokeAgent`, `authorizeAgent`) reverts `AgentCannotCallWallet`; arbitrary contract calls outside the protocol allow-list are blocked. The agent can move only ETH and allow-listed tokens to allow-listed recipients.

Test battery — forge test: 264 passed, 0 failed

SUITE	TESTS	FOCUS
<code>BVCCAgentWallet.t.sol</code>	50	Agent permission core: limits, budgets, expiry, pause, batches
<code>UniversalRouterValidator.t.sol</code>	42	Router corpus, attacker recipients, fuzz over all 256 command bytes

BVCCAgentWalletAdvanced.t.sol	40	Fee accounting, approve regression tests, edge cases
CallPolicyAdversarial.t.sol	34	Policy bypass attempts, registry governance, malformed calldata
BVCCPositionManagerValidator.t.sol	26	Uniswap v4 LP calldata validation
BVCCWallet.t.sol	23	Base smart account: execution, fees, guardian recovery
BVCCWalletFactory.t.sol · BVCCAgentWalletFactory.t.sol	18	CREATE2 factories, deterministic addresses, kill switch
BVCCHookRegistry.t.sol	11	Hook approval registry and its timelock
AaveIntegration.t.sol	10	Forked-mainnet Aave cycle and adversarial variants
BVCCPMValidatorSdkVectors.t.sol · ERC7739*.t.sol · Create2Consistency.t.sol · GasBench.t.sol	10	SDK golden vectors, ERC-7739 signatures, address freeze guard, gas

Adversarial policy tests (CallPolicyAdversarial.t.sol): Aave withdraw/supply/borrow/repay with an external to OR onBehalfOf revert PinnedArgMismatch; an unregistered selector reverts SelectorNotAllowed; PIN_PROTOCOL rejects a spender that is not whitelisted; a DEEP call denies when the validator returns false, reverts, has no code or is not registered, and passes only on true; the 48-hour timelock, freeze and cancel paths behave as specified; policy = 0 revokes; short calldata, out-of-range pins and dirty high bits are all rejected; and a batch containing one bad item reverts as a whole.

Aave on a mainnet fork (AaveIntegration.t.sol): a full supply → borrow → repay → withdraw cycle, with the 0.15% agent fee charged exactly on withdraw and borrow and not at all on supply and repay; budget consumption verified through the approve path; an external beneficiary on any of the four calls reverts; flashLoanSimple and an unregistered setUserEMode revert; a Pool missing from allowedRecipients reverts; and the approve+supply batch rolls back atomically on failure.

The permission layer had been attacked from every angle across three rounds; guardian recovery and the signature path had not. Both hold funds, so both were given the same treatment: twenty tests written to make something work that must not (test/RecoveryAdversarial.t.sol, test/SignatureAdversarial.t.sol).

14.1 Signature path — no findings

Ten attacks against the ERC-7739 / WebAuthn path, all rejected. The two that matter most are the replay pairs, because this wallet's design invites both: the address is identical on every network, and one passkey backs a smart wallet and an agent wallet at two different addresses.

ATTACK	RESULT
Replay a signature from one chain onto the same address on another	Rejected chainId is bound in the verifier's domain
Replay a signature between the owner's own smart and agent wallets	Rejected verifying contract is bound
Malleated signature (s replaced by N - s)	Rejected

Zeroed signature	Rejected
Signature over different contents, or another dApp's domain	Rejected
Tampered authenticatorData (user-verification flag dropped)	Rejected
Challenge swapped for another digest	Rejected
webauthn.create presented where webauthn.get is required	Rejected

On-chain verification — Arbitrum One (mainnet)

Test wallet `0x605A4C65dBa64bd26d9F7701e1e4Cda48c594A5c`, agent `0x38529C66F3cf22453D66B9E2A20FdF2676544aB4`.

DATE	OPERATION	RESULT
2026-07-16	Agent swap USDC → WETH through SwapRouter02, driven by the MCP server <code>0x4768603d38b03f9823d5d25e53fb03970b8e4d294ebee791d0ae9ef92e71a8ae</code>	OK USDC debited exactly, WETH to the wallet, fee 0.15% exact
2026-07-20	Agent swap through the Universal Router — an atomic 3-call batch crossing three different layers <code>0x6f831b37ae5db240256bf6bf99c8ce418edb75a1c38e37d6ed3e2c3a052a3bc0</code>	OK gas 419,124 · USDC -0.100000 · fee 0.000000078468831120 WETH = 0.15% exact · every recipient the wallet
2026-07-20	Same swap with recipient = attacker (simulated, no funds moved)	DENIED reverts before executing

The three calls of that batch each traverse a different guard: `USDC.approve(Permit2)` is case 2b and is checked against `allowedRecipients`; `Permit2.approve` is case 3 with `PIN_PROTOCOL` on the spender; `UniversalRouter.execute` is case 3 with `DEEP` and is decided by the validator.

Gate behaviour, measured before and after the validator's 48-hour activation:

<code>REGISTRY.VALIDATE(...)</code>	BEFORE ACTIVATION	AFTER ACTIVATION
Legitimate swap (recipient = wallet)	false	true
Theft attempt (recipient = attacker)	false	false
Command <code>0x40</code> (cross-chain bridge)	false	false
Target ≠ the bound router	false	false

The mask fix is therefore confirmed on mainnet, not only in tests, and the fail-closed property is visible in the "before" column: until the validator was active, even the legitimate swap was denied.

Gas

OPERATION	GAS
<code>createWallet</code>	4,548,977
<code>authorizeAgent</code>	166,803
<code>setCallPolicy</code>	25,912
<code>executeAsAgent</code> (case 3, policy checked)	64,758

C Static Analysis

Slither was run over the tree at each round. This appendix records the results and, as important, which results are false positives and why — so they are not re-investigated, and so the two virtual functions flagged as dead code are not deleted.

Slither 0.11.3 was run on both repositories prior to redeploy. It reported 39 results across 59 contracts; **none were actionable**. The table summarizes the categories and their disposition.

CATEGORY	EXAMPLES	DISPOSITION
By-design patterns	<code>block.timestamp</code> comparisons, inline assembly, low-level calls, calls-in-loop (batch)	Accepted
False positives	“uninitialized” mapping; reentrancy in <code>createWallet</code> (only an emit after trusted <code>setGuardians</code>)	No action
Dead code — <code>_feeNumerator()</code>	Flagged as unused	False positive — verified in use (see note)
Missing zero-check — <code>factory owner_</code>	Constructor sets owner without validation	Fixed
Cosmetic	Naming, literal digits, cyclomatic complexity	Accepted

Verification note — tool findings are not facts. Slither marked `_feeNumerator()` as removable dead code. A source check showed it **is** used to compute the protocol fee (base 0.05% / agent override 0.15%); removing it would have broken fee logic. The finding was a false positive caused by virtual-function dispatch and was **not** acted upon.

Remediation applied

A defensive zero-address check — `if (owner_ == address(0)) revert ZeroOwner();` — was added to both factory constructors, synchronized across both repositories. Full suite green: **forge test = 128/128** in each repo. Slither was re-run on the V2 contracts after the gas hardening (Appendix D): **0 results**; the unit suite now stands at **131/131**.

Static analysis — Slither re-run on V3

Slither 0.11.3 was re-run over the V3 tree on 2026-07-27 (`slither . --exclude-dependencies`, 65 contracts, 100 detectors): **54 results — 3 High, 29 Low, 22 Informational**. Every High-impact result was traced back to the source before being accepted or dismissed. One of them is real: the reentrancy-eth result is a confirmed HIGH finding, reproduced with a proof-of-concept test and still open at the time of writing (BVCC-01).

Read this run, not the V2 one. The V2 pass reported “0 results”, but only because the analyzer could not lower OpenZeppelin’s transient-storage `ReentrancyGuard` to IR and bailed out (Appendix C). That zero was the tool giving up, not a clean bill of health. The V3 tree analyzes end to end — the detectors do reach `executeAsAgent`, as the reentrancy result below shows — so this is the static-analysis datapoint that carries weight, and its result classes line up with the 39 results of the V1 run.

DETECTOR	TARGET	VERDICT
reentrancy-eth confidence: medium	executeAsAgent writes _currentAgent = 0 after super.execute()	CONFIRMED — HIGH — reproduced with a PoC test (BVCC-01)
uninitialized- state ×2 confidence: high	_tokenTotalSpent, _tokenDailySpent	False positive — both are mappings, which have no initializer and default to zero

D Previously Remediated — V1 & V2

These findings were reproduced and closed in generations that V4 supersedes. They are retained with their evidence because wallets created by those factories are still live until their owners migrate, and because the remediations they describe are inherited by V4.

HIGH

FIXED & REDEPLOYED

Description

An agent's per-token **daily** and **total** budgets could be bypassed entirely. The agent validation accounted spend for transfer but charged approve only against the per-transaction cap — not the daily or total budgets, and without restricting the spender. An agent could therefore approve an allowance (only per-tx-capped) and drain it with an external `transferFrom`, never touching the cumulative counters.

Proof of concept (executed on-chain, budgets exhausted at daily 2/2 · total 2/3)

STEP	ACTION	RESULT
1	<code>executeAsAgent</code> → <code>USDC.approve(B, 1)</code>	Passes — per-tx OK, daily ignored
2	As EOA: <code>USDC.transferFrom(wallet, B, 1)</code> (outside <code>executeAsAgent</code>)	1 USDC extracted; counters unchanged

A plain `transfer` of the same amount correctly reverted `TokenDailyLimitExceeded`; `approve` did not. Because `approve` had no daily limit, the `approve(1) → transferFrom(1)` cycle was repeatable — draining the wallet past both the daily (2) and total (3) caps. The PoC funds were returned to the wallet.

Remediation

In the `executeAsAgent` accumulation loop the condition was extended with `|| _isApprove(batch[i].callData)` so that `_applyTokenSpend` charges approve against the same daily/total caps and updates the counters. The choice is deliberately conservative (an `approve` overwrites the allowance, so it may over-count — the safe direction). Six regression tests were added; **5 fail without the fix, all pass with it.**

Bounded surface. Tokens accept only `transfer` and `approve` by selector; `transferFrom` and `increaseAllowance` are blocked. ETH has no delegation primitive, so it is unaffected. `approve` was the sole vector — now closed and re-verified post-redeploy (see Appendix B, test #4 context).

Both factories changed bytecode (zero-check), producing new deterministic CREATE2 addresses on Arbitrum Sepolia. The frontend network configuration was updated accordingly.

CONTRACT	NEW ADDRESS (PATCHED)	PREVIOUS (DEPRECATED)
AgentWalletFactory	0xc87aa10747A92B472EF6B36e190B84c897a2953e	0xe7ad...7e04
SmartWalletFactory	0xa5290A51a73903176e09C864E1542a07da67BD12	0x6890...F58d

Privilege separation

- **Deployer (no control):** 0xA3e06B6443D911915bb4eD157832d43C25D6617D — the factory's owner is set from a constructor argument, not `msg.sender`, so the deployer holds no privileges.
- **Admin / kill-switch owner:** 0x3145Bd5e2489d8bDdAb17F23F26F07Ac5aD55A4c — immutable; the only privileged action is the one-way `kill()`.

Runtime sizes within the EIP-170 limit (AgentWallet 21,169 B; AgentWalletFactory 23,803 B — 773 B of headroom).

The base wallet supports social recovery: a 2-of-3 guardian quorum can rotate the owner passkey after a timelock. This was exercised end-to-end on wallet

0xEBC851bA9e8Ce32A2cb8eB99544F5F024f06E61d and is now **complete**.

Process

- **Initiation (2026-06-06).** Recovery proposed toward a new owner passkey; guardians 1 and 2 (0xa337...5908, 0x8e06...2Ccc) approved — quorum 2/2.
- **Timelock.** On-chain `recoveryReadyAt` = 2026-06-08 17:24 UTC; throughout the window the owner retained the ability to cancel.
- **Execution (2026-06-08).** After the timelock elapsed, `executeRecovery()` was called from the wallet /recover UI. The on-chain signer rotated to the new passkey (X 0x303591... / Y 0xb858ed...), replacing the previous owner (X 0x070f7a...).
- **Positive check.** The new owner immediately signed a real **19 USDC** transfer with the new Face ID — confirmed on-chain.

Negative test — the old passkey is neutralized

To confirm the rotation actually revoked the previous key, a transfer was attempted with the **old** passkey, whose WebAuthn credential is still present in the test authenticator. The signature is produced locally but must be validated on-chain against the current owner.

#	ACTION	EXPECTED	RESULT
1	Send 0.0001 ETH, UserOp signed by the OLD passkey	rejected at validation	FailedOp(0, AA24 signature error)
2	On-chain effect (ETH / USDC balances & nonce)	no change	Unchanged

Result. Recovery completed successfully: ownership moved to the new passkey, the new key operates the wallet, and the old key is cryptographically rejected by the EntryPoint during signature validation — it can no longer move funds even though its local credential still exists.

HIGH (availability / overpayment)

FIXED — V2 DEPLOYED 2026-06-11

Description

On Arbitrum mainnet, swaps through the wallet failed at gas estimation. The Case-3 fee logic scans calldata for token candidates and probes each one with `balanceOf`. Arbitrum system precompiles (0x64-0x6f) report bytecode (0xfe), so the code-length guard did not filter them — and calling them consumes **all forwarded gas** instead of reverting cheaply. The estimator kept raising the gas limit until ~32M so the probe would “fit”; that inflated limit is what the wallet would have prefunded (a 64× overpayment risk), and in practice the operation failed.

	BEFORE (BUG)	AFTER (V2 FIX)
Frontend / bundler gas estimate	~32,000,000	~400,000
Gas-burning candidate effect	burns all forwarded gas	capped at 100k, execution continues

Outcome	swap fails (or 64× overpayment)	normal swap, ~\$0.02–0.05 on Arbitrum
---------	---------------------------------	---------------------------------------

Remediation (V2)

- **Gas-capped probes.** A shared constant `PROBE_GAS_CAP = 100_000` caps every `balanceOf` staticcall in both `_tryBalanceOf` (snapshot) and `_collectFeesOnIncrease` (fee collection). Worst case across a full scan is bounded to ~2.2M gas vs 32M before; real ERC-20s need well under the cap, so fee accounting is unaffected.
- **Fee collection can never break a user call.** If a token misbehaves after the swap, its fee is skipped instead of reverting the user's transaction.
- **Explicit accounting bound.** `executeAsAgent` now requires `totalEth ≤ type(uint128).max` before casting, removing a silent-truncation edge case for agents configured with no ETH limits.

Verification

- **forge test = 131/131** — regression tests added: a gas-burning candidate does not starve the swap; a reverting `balanceOf` is filtered non-fatally; a fat-return token is still snapshotted.
- **Slither 0.11.3 re-run on V2: 0 results.** (Tool note: Slither cannot generate IR for OpenZeppelin's transient-storage `ReentrancyGuard` internals — a known parser limitation, not a finding.)
- **EIP-170 sizes:** `AgentWallet` 21,214 B; `AgentWalletFactory` 23,848 B (728 B headroom); `SmartWallet` 13,432 B; `SmartWalletFactory` 16,030 B.

V2 deployment — deterministic CREATE2, same address on every chain

CONTRACT	ADDRESS (IDENTICAL ON ALL NETWORKS)
SmartWalletFactory V2	0x230b7010529AB6977Dd8581B3eF018ef865BdEf1
AgentWalletFactory V2	0x8D9e24022777173AD6336e00884b6C87c7EF054c

Deployed 2026-06-11 to Arbitrum One (42161), BNB Chain (56) and Arbitrum Sepolia (421614). Wallets created by pre-V2 factories do not receive the fix — new wallets must be created through the V2 factories.

Three issues in the wallet web application were fixed and synchronized to the public repository.

AREA	ISSUE	FIX
lib/wallet.ts	Credential recovery queried only the base factory event, so re-opening an <i>agent</i> wallet by address failed (“no active wallet”).	Also query <code>agentFactory</code> / <code>AgentWalletCreated</code> .
Agents page	Per-token limits shown as text only.	Read <code>getTokenSpent</code> on-chain; render Daily/Total progress bars (verified live).
Deploy flow	App-supplied gas under-estimated the ~21 KB agent-wallet deploy on Arbitrum, causing signing failures.	Defer to the wallet’s network estimate.

E Remediation Timeline

ROUND	DATES	FOCUS	OUTCOME
V1	2026-06-06 → 06-08	Agent permission model, first static analysis	BVCC-A1 found, fixed, redeployed; permission battery and guardian recovery verified end-to-end
V2	2026-06-11 → 06-12	Mainnet gas behaviour	BVCC-A2 found and fixed; deterministic multichain deployment across six networks
V3	2026-07-21	Call-policy layer and validator registry	Fund-exfiltration path closed; BVCC-02 and BVCC-12 caught inside the work, before deployment
V4	2026-07-27 → 07-28	Post-release review, second tooling pass	BVCC-01, BVCC-04 to BVCC-08 and BVCC-11 found and fixed; deployed and verified on six networks
—	2026-07-28 → 07-29	Validator activation	Deep-validation layer completed on all six networks; v4 liquidity cycle exercised on mainnet

The V3 deployment record, build freeze, off-chain layer and migration notes follow.

V3 deployment — deterministic CREATE2

CONTRACT	ADDRESS (IDENTICAL ON ALL SIX NETWORKS)
SmartWalletFactory V3	0xD42F61AA856A4f47885Ecd2D0ce119411d53C192
AgentWalletFactory V3	0xd866a7563cDaC9F71423be3332b62c329C676064
ValidatorRegistry	0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2
HookRegistry	0x551C6e7ABdA04a110790888e711198f25621b066

Networks: Arbitrum One (42161), Ethereum (1), BNB Chain (56), Base (8453), Polygon (137) and Arbitrum Sepolia (421614). All four contracts were re-checked on 2026-07-27 through the explorers' API: present, byte-identical and source-verified on all six chains.

Universal Router validators are bound to one router each, so their addresses differ per network:

NETWORK	UR VALIDATOR	REGISTRY STATUS
Arbitrum One	0xD1522594e185C882bfDEc6FFD4DE8a53F69AF0Ca	Active
Ethereum	0x3615a0Fc0663C0201B543deC9816d5Ef30a82ffb	Active
BNB Chain	0x6945f861e0D7FCD566739000f3fDC1D44896D815	Active
Base	0x4A2Fc73AADEfC84513defaa0c444d2150AC6A6D8	Active
Polygon	0xC1C584E4149eD2EF20DA0155b1B8257232102fF	Active
Arbitrum Sepolia	0x87550034627966e32d82E3b8AdB3f19c62E8d86E	Active

Status read directly from `validators(router)` on each chain, last on 2026-07-28 after Arbitrum Sepolia was activated. A deployed-but-unactivated validator is inert in the safe direction: the corresponding DEEP calls fail closed. Per-chain provenance manifests (addresses plus router and validator codehashes) live in `contracts/deployments/`. The Uniswap v4 `PositionManager` validators were activated on all six networks in the same round — see Appendix A for the addresses and the per-network binding check, and Appendix D for the liquidity cycle run against them.

Build freeze

Deterministic CREATE2 addresses require a frozen build: solc **0.8.36**, `via_ir`, `optimizer_runs = 50`, EVM `cancun`. That compiler was chosen deliberately to stay clear of the 0.8.28–0.8.33 `via-IR` pipeline bug. The agent factory's runtime size is **24,283 bytes**, leaving 293 bytes of headroom under the EIP-170 limit — any future addition must be measured against it.

Off-chain layer — SDK and MCP 0.2.0

The client libraries were made policy-aware so that the on-chain guard is not discovered only at broadcast time. `getCapabilities` now gates a swap on the registered policies *and* the validator registry rather than on the permission struct alone, and reports the wallet version plus any missing policies; the four V3 errors decode into actionable messages, distinguishing "no validator active" (do not retry) from "the validator rejected this calldata" (fixable). `setCallPolicy` is deliberately **not** exposed by the SDK: policies are written by the owner's passkey, never by an agent's key. Aave lending and its planners ship through the same catalog, with the beneficiary always pinned to the wallet.

Migration

V3 wallet bytecode differs from V2, so the CREATE2 addresses differ: existing users create a new wallet and move their funds, exactly as in the V1 → V2 migration, after which the V2 factories are killed. Future capabilities — registering further validators, adding protocol presets — are bytecode-neutral and will not force another migration.

F Key Identifiers & Addresses

ITEM	VALUE
Network	Arbitrum Sepolia (421614)
Patched wallet	0x1ED261BA084c4E6117daf289f3ca233BBA3cF0C3
Agent (role B)	0x38529C66F3cf22453D66B9E2A20FdF2676544aB4
AgentWalletFactory (Sepolia, pre-V2)	0xc87aa10747A92B472EF6B36e190B84c897a2953e
SmartWalletFactory (Sepolia, pre-V2)	0xa5290A51a73903176e09C864E1542a07da67BD12
SmartWalletFactory V2 (all networks)	0x230b7010529AB6977Dd8581B3eF018ef865BdEf1
AgentWalletFactory V2 (all networks)	0x8D9e24022777173AD6336e00884b6C87c7EF054c
SmartWalletFactory V3 (all networks)	0xD42F61AA856A4f47885Ecd2D0ce119411d53C192
AgentWalletFactory V3 (all networks)	0xd866a7563cDaC9F71423be3332b62c329C676064
SmartWalletFactory V4 (all networks)	0xfd105197109244483b5f870501326E6faec9F93c
AgentWalletFactory V4 (all networks)	0xf3A61F9d64d45362E149A111289546523BCd26a6
ValidatorRegistry (all networks, V3 & V4)	0x5e371D54AC97a57B0a99145Ed04A3c9fA07850C2
HookRegistry (all networks, V3 & V4)	0x551C6e7ABdA04a110790888e711198f25621b066
v4 LP test position (Arbitrum One)	tokenId 190877 – minted and burned by an agent
V3 test wallet (Arbitrum One)	0x605A4C65dBa64bd26d9F7701e1e4Cda48c594A5c
Governance admin (hardware wallet)	0x3145Bd5e2489d8bDdAb17F23F26F07Ac5aD55A4c
V2/V3/V4 networks	Arbitrum One · Base · BNB Chain · Ethereum · Polygon · Arbitrum Sepolia
V3/V4 build	solc 0.8.36 · via-IR · optimizer 50 · EVM cancun (frozen for CREATE2)
Tooling	Slither 0.11.3 · Foundry (forge 1.5.1) · viem 2.x

Confidential — Block Venture Chain Capital. Prepared for internal review and repository inclusion. The V1 permission battery was executed on a public testnet (Arbitrum Sepolia); the V3 call-policy layer and the v4 liquidity cycle were additionally verified on

Arbitrum One mainnet. V2, V3 and V4 factories are deployed on Arbitrum One, Ethereum, BNB Chain, Base, Polygon and Arbitrum Sepolia, with the Universal Router and v4 PositionManager validators registered on all six.

G Process Notes

Recorded because a review that only lists what it found overstates its own reliability.

Three lessons are worth recording alongside the findings. The reentrancy was dismissed once, on plausible reasoning about EOAs, before a proof-of-concept contradicted it: assumptions about the shape of an address deserve a test, not an argument. The most serious issues here were each found by looking again at code that had already passed a review — guardian squatting came from a pass with different tooling, credential rotation from a follow-up of that same pass, and two more from attacking recovery directly. And the static-analysis count moved by one across all of it: it did not rise while the reentrancy was exploitable for three generations, and it did not fall when the fix closed it. Tools find patterns; the findings here came from reading code and asking what an attacker would do with it. The contracts remain internally reviewed and are not externally audited.