SECURITY AUDIT

Greek.fi

Onchain options + Bebop integration

FINAL REPORT

Oct 2026

BAILSEC.IO

Disclaimer:

BailSec was engaged by the client to perform a time-boxed technical security assessment of the target identified in this report. The assessment applies only to the scope, version, commit, deployment, materials and information identified in the report. Its findings are point-in-time and not exhaustive. Changes made after the assessment—including remediation changes—were not assessed unless expressly stated otherwise and may affect the applicability of the report's conclusions.

No security assessment can identify every vulnerability or guarantee that a system is secure, error-free or protected against attack or loss. Only the components, assumptions, threat models and risk categories expressly identified as in scope were assessed. The report should be read in full, including its scope, exclusions, unresolved findings and remediation status. The client remains responsible for design, testing, remediation, deployment, operation, monitoring and incident response.

This report is a technical assessment, not a certification, attestation, investment recommendation, or legal, financial, regulatory or tax advice. It does not constitute an endorsement of the client, the assessed system or any related project.

This report was prepared for the client under a separate engagement agreement. If published, its public availability is for transparency and informational purposes only. To the fullest extent permitted by applicable law, publication does not create an auditor-client relationship, duty of care or right of reliance in favor of any third party. Other readers should conduct their own diligence and obtain appropriate professional advice.

Only the complete, unmodified report as issued by BailSec is authoritative. The signed engagement agreement exclusively governs the contractual rights and obligations between BailSec and the client, including warranties, remedies, liability limitations, indemnities and dispute resolution. Nothing in this notice excludes liability that cannot lawfully be excluded.

1. Project Details

Important: Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Greek.fi – Onchain options + Bebop integration – Audit Report
Website	greek.fi
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/greekfi/contracts/tree/c7e2027c87a22c851062a94073ece0e8d0af4b88 Option.sol Receipt.sol Factory.sol Permissions.sol lib/clone (submodule of https://github.com/greekfi/clone @ aafc4a8a4066b41a9e90cd7580ff5f3eba138f1e): ClonesWithImmutableArgs.sol Clone.sol Integration reference: https://github.com/greekfi/contracts/blob/main/test/BebopE2E.t.sol (BebopSettlement 0xbbbbbBB520d69a9775E85b458C58c648259FAD5F on Ethereum)
Resolution 1	https://github.com/greekfi/contracts/tree/c7200d4d3b9053bf820f0b495dbeec9eb630c6b0 lib/clone (submodule of https://github.com/greekfi/clone @ aafc4a8a4066b41a9e90cd7580ff5f3eba138f1e)

Resolution 2	https://github.com/greekfi/contracts/tree/1ec4066e9ed39e91452f0f262f297e6955fbd2a9 lib/clone (submodule of https://github.com/greekfi/clone @ 13741b2cf3ec39dbdc6b5298bfb3006fa448ca4e)

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High						
Medium	1	1				
Low	2	2				
Informational	16	4		12		
Governance	2			2		
Total	21	7		14		

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

3. Detection

Factory

The Factory deploys option markets and is the protocol's shared allowance and permission hub. In its constructor it deploys two templates: a Receipt template (`RECEIPT_CLONE`) and an Option template (`OPTION_CLONE`). Each market consists of an EIP-1167 clone of the Option template and a clone-with-immutable-args of the Receipt template that carries 112 bytes of market parameters in its runtime code.

Market creation is permissionless. Users approve their tokens once to the Factory; Factory-created Receipts then pull collateral (on mint) and consideration (on exercise) through `Factory.transferFrom`. The Factory also stores the per-owner, per-operator permission bitmasks (`Perm`) that every market reads, the global action-pause mask (`Pause`), and the default redeem fee copied into each new Receipt.

Appendix: Market Creation

`createOption(params)` is `createOption2(params, 0, 0)`. `createOption2` validates the parameters (pause flag `CREATE`, `strike != 0`, distinct non-zero tokens, `expirationDate` in the future, `windowSeconds > 0` for European markets, both `decimals() <= 36`, and `strike <= type(uint256).max / 10**(consDec - collDec)` when the consideration has more decimals).

Markets are registered by `optionKey` = hash of (collateral, consideration, expirationDate, strike, isPut, isEuro, windowSeconds): if the key exists the existing Option is returned (get-or-create). Otherwise the Option clone is deployed (CREATE, or CREATE2 with a salt namespaced by `msg.sender`), then the Receipt clone with its immutable args, the Receipt fee is set from `Factory.feeBps`, `Option.init(receipt)` wires the pair, and the Receipt is registered in `receipts[]`. `createOptions` / `createOptions2` are batch versions.

Appendix: Allowance Hub

`transferFrom(from, to, amount, token)` can only be called by a registered Receipt. It pulls `amount` of `token` from `from` to `to` and reverts with `FeeOnTransferNotSupported` if the receiver's balance grew by less than `amount`. Receipts only reach it from `mint` and `exercise`,

which are themselves only callable by the paired Option and gated by the Option's permission checks.

Appendix: Permission Registry

`permissions[owner][operator]` is a bitmask of `Perm` bits (`TRANSFER`, `MINT`, `BURN`, `REDEEM`, `EXERCISE`, `TRANSFER_RECEIPT`). It is written by `setPermissions` / `addPermissions` (the caller's own row) and by `setPermissionsFor` / `addPermissionsFor` (any row whose account is the caller, or whose `Ownable(account).owner()` is the caller). A grant applies to every market of the Factory, current and future.

The self entry `permissions[a][a]` opts the account into transfer-time automation: `MINT` = auto-mint on a transfer shortfall, `BURN` = auto-burn of matched pairs on transfers it receives and initiates.

Appendix: Pauses and Fees

The owner can pause and unpause any combination of `MINT`, `EXERCISE`, `REDEEM`, `BURN`, `TRANSFER`, `CREATE` and `EXPIRE` instantly; the mask applies to every existing market.

`setFee` sets the default redeem fee for markets created afterwards (at most 1000 bps). Ownership cannot be renounced.

Privileged Functions

- pause / unpause
- setFee
- transferOwnership
- setPermissionsFor / addPermissionsFor

Invariants

- Only Receipts created by this Factory can pull user funds through `transferFrom`.
- A pull must credit the Receipt with at least the requested amount.
- At most one market exists per `optionKey`.
- A market cannot be created with a zero strike, identical or zero tokens, a past expiration, a zero European window, more than 36 decimals, or a strike that overflows the conversion.
- Every new Receipt starts with the current `Factory.feeBps`, which never exceeds 1000 bps.

- Only the owner can pause, unpause or change the default fee; ownership cannot be renounced.
- A permission row can only be written by its account (or its `owner()`), and only with bits inside `Perm.ALL`.
- CREATE2 salts are namespaced by the caller, so a mined address cannot be taken by another direct caller.

Issue_01	Global pause can end exercise rights on every market
Severity	**Governance**
Description	The Factory owner can pause any combination of actions (`MINT`, `EXERCISE`, `REDEEM`, `BURN`, `TRANSFER`, `CREATE`, `EXPIRE`) instantly and without a timelock. The pause applies to every market the Factory has created. No deadline moves while an action is paused, because `expirationDate` and `exerciseDeadline` are immutable arguments of each Receipt clone. This has the following effects on users: a) A pause of `EXERCISE` that spans a market's deadline ends the exercise right for good. For example, an 8-hour European window that falls inside a pause can never be exercised. As soon as `block.timestamp > exerciseDeadline`, the writers can redeem the full collateral (if `REDEEM` is not paused), so the in-the-money value of every long holder moves to the writers. A pause of `TRANSFER` has the same effect on longs held in a vault or an AMM, which must be withdrawn before they can be exercised. b) Pausing `BURN`, `REDEEM` and `TRANSFER` together freezes every exit of the short side, including pair-burn, which the documentation presents as the "guaranteed exit" for writers. The pause flags were added to a design in which transfers, pair-burn and exercise could never be stopped. `IFactory` and `context/architecture.md` state that a pause does not extend deadlines, but `README.md` still says that "The core has no oracle, upgrade hook, pause switch, or emergency withdrawal", and its trust boundaries do not list the pause power.
Recommendations	Consider restricting the pause to actions that open new positions (`MINT`, `CREATE`) and never pausing the exits (`EXERCISE`, `REDEEM`, `BURN`, `TRANSFER`), so a pause can never take value from existing

	holders. If pausing exits must stay, consider using a multisig as owner, ensuring that the signers are trusted entities, never pausing `EXERCISE` close to a market's `exerciseDeadline`, and updating the README trust model.
Comments / Resolution	Acknowledged.

Issue_02	`setPermissionsFor` allows the `owner()` of any contract to take over that contract's Greek positions
Severity	Medium
Description	A permission row in the Factory can be written not only by the account itself, but also by whoever `Ownable(account).owner()` returns. Exposing `owner()` does not mean that the owner controls the funds a contract holds. For every contract that exposes `owner()` and holds Greek tokens, this check gives the owner full control over those positions, even when the contract's own code never allows it. Examples are Morpho Blue (owner = Morpho governance) holding Receipts as collateral, as in the repository's experimental Morpho integration, and vaults whose owner is an admin. The owner grants an operator `TRANSFER` / `TRANSFER_RECEIPT` and moves every Greek token deposited by users. Where the custodian has approved collateral to the Factory (for example a vault that writes options), `MINT` also mints against that allowance. Grants are not cleared when `owner()` changes. They are visible through `PermissionsUpdated` events, but a previous owner can use them before the new owner revokes them. The behaviour is documented in the `IFactory` NatSpec, but the users who lose funds are the depositors of third-party contracts, who do not consent to it. The same applies to EOAs delegated through EIP-7702: if the delegate code exposes an `owner()` that returns another address, that address can write the EOA's permission row. This contradicts `IFactory.sol:287`, which states that "An EOA target can only manage its own row". This also contradicts `IReceipt.sol:284-285`, which states that a Receipt held in a vault or Morpho market cannot be force-redeemed by an unauthorised third party: the `owner()` of such a contract can grant `REDEEM` or `TRANSFER_RECEIPT` over it.

Recommendations	Consider removing the `owner()` branch so that only the account itself can write its permissions. Contracts that need to grant permissions to a third party can call `setPermissions` themselves.
Comments / Resolution	Resolved. setPermissionsFor and addPermissionsFor were removed, so an account can only set its own permissions.

Issue_03	Single-step ownership transfer can permanently lose the Factory owner role
Severity	Informational
Description	The Factory inherits OpenZeppelin's single-step `Ownable`: `transferOwnership` assigns the new owner immediately, without the new owner accepting it, and only rejects `address(0)`. `renounceOwnership` is disabled precisely because an unreachable owner would break the protocol, but a transfer to a mistyped or uncontrolled address leads to the same state and cannot be undone. The owner role is required to operate every market: - `pause` / `unpause`: if any action is paused at the time ownership is lost, it can never be unpaused. With `EXERCISE`, `BURN`, `REDEEM` or `TRANSFER` paused, the exits of every market stay frozen while their exercise deadlines pass. - `Receipt.setFee` and `Factory.setFee`: fees can no longer be changed, including lowered. - `collectFees` pays the accrued fees of every Receipt to `factory.owner()`, so they go to the wrong address, and `sweep` is no longer reachable.
Recommendations	This can also be acknowledged, as it is just an informational note
Comments / Resolution	Acknowledged.

Issue_04	Fixed exercise deadline has no grace period for L2 sequencer outages
Severity	**Informational**
Description	Exercise closes strictly at `exerciseDeadline`, and the window length is chosen by the market creator (European markets only need `windowSeconds > 0`, and the frontend suggests 8 hours). According to the deployment documentation, the protocol is deployed on single-sequencer L2 rollups (Optimism, Base, Arbitrum, Hemi and Robinhood Chain). If the sequencer is down for the whole window, long holders cannot exercise. Forced inclusion through L1 takes up to about 12 hours on Optimism, Base and Hemi, up to about 24 hours on Arbitrum and up to about 4 days on Robinhood Chain (where a transaction filterer can also block forced transactions), which is longer than the suggested window. Once the deadline passes, the writers redeem the full collateral. The documentation only covers sequencer downtime for the oracle, not for the exercise window.
Recommendations	Consider documenting the chain-specific liveness risk for writers and buyers, and using longer default windows on L2 deployments that exceed the chain's forced-inclusion delay.
Comments / Resolution	Acknowledged. The default window was shortened to 1 hour and the liveness risk is documented in the windowSeconds NatSpec.

Issue_05	Fee cap is a magic number that must not exceed the Receipt's cap
Severity	Informational
Description	`Factory.setFee` caps the default fee with the literal `1000`. Every new Receipt is initialized with this fee at creation, and the Receipt enforces its own cap with a separate literal. If the Factory cap is ever raised above the Receipt cap, every market creation reverts while the higher fee is set.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

Issue_06	Token decimals are validated on one read and stored from a second read
Severity	Informational
Description	`createOption2` reads `decimals()` of both tokens and validates them: each must be at most 36, and the strike must not overflow the conversion when the consideration has more decimals. `_receiptArgs` then calls `decimals()` on both tokens a second time and stores these unvalidated values in the Receipt's immutable arguments, which every conversion uses. A token whose `decimals()` returns a different value on the second read can therefore create a market whose stored decimals never passed the checks. Depending on the value, `toConsideration` either converts at a scale wrong by powers of ten, so exercise prices become wrong, or overflows and reverts, which blocks exercise and the consideration leg of redemption for the whole market. The stored decimals are also never refreshed. If a token later changes its `decimals()` (for example through an upgrade), the Receipt keeps converting with the old value, while `Option.decimals()` reads the token live, so the Option and Receipt report different decimals for the same market.
Recommendations	Consider acknowledging this issue, as it is merely an informational note.
Comments / Resolution	Acknowledged.

Issue_07	European markets can be created with an exercise window no block can reach
Severity	Informational
Description	For European markets, `createOption2` only requires `windowSeconds > 0`, so a creator can set a window of a single second. European Options can only be exercised from `expirationDate` to `exerciseDeadline = expirationDate + windowSeconds` (both inclusive), so a 1-second window allows exercise at only two timestamps. On Ethereum, blocks are produced at fixed 12-second slots. Any window of about 10 seconds or less can fall entirely between two block timestamps, so no transaction can ever exercise these Options. Even when a block lands inside the window, the exercise has to be included in that exact block. After the deadline, the writers redeem the full collateral, so the long holders receive nothing. Such a market looks like any other European market, because the market name does not show the window, so a buyer only notices by reading `exerciseDeadline`. The window length is permissionless by design, but the NatSpec does not warn that a window can be too short to be usable.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

Option

The Option contract is the long side of a market. One Option is the right, not the obligation, to receive one unit of collateral by paying `toConsideration(amount, roundUp)` of the consideration token. Each market's Option is an EIP-1167 clone of `OPTION_CLONE`, initialized once by the Factory with its Receipt; the template itself is locked with a `0xdead` sentinel. Its `decimals()` are the collateral's decimals, read live. The name and symbol are rendered as `OPTA-<coll>-<cons>-<strike>-<YYYY-MM-DD>` (`OPTE` for European), with the strike inverted back to human form when `isPut` is set.

Appendix: Mint and Pair-Burn

`mint(amount)` (self) and `mint(account, amount)` (operator with `MINT`) deposit `amount` collateral into the Receipt via the Factory and mint `amount` Receipts and `amount` Options to the account. Minting is only possible strictly before `expirationDate`.

`burn(amount)` / `burn(account, amount)` (operator with `BURN`) burn equal amounts of both tokens and return the collateral 1:1 without a fee, at any time.

Appendix: Transfer Hooks

`transfer` and `transferFrom` run two optional legs around the ERC-20 transfer:
- **auto-mint**: if the sender's balance is short and `permissions[from][msg.sender]` has `MINT`, the shortfall is minted from `from`'s collateral allowance first;
- **auto-burn**: if `permissions[to][msg.sender]` has `BURN`, `min(receiptBalance(to), value)` pairs of `to` are burned after the transfer and the collateral returned to `to`.

`transferFrom` skips the ERC-20 allowance when the caller holds `TRANSFER` in `from`'s row.

Appendix: Puts

There is no **PUT** logic in the code. A put on WETH at 3,000 is a market with USDC as collateral, WETH as consideration and `strike = 1e36 / 3000e18`; exercising it delivers WETH and receives USDC. `isPut` is a label that only inverts the displayed strike and is part of `optionKey`.

Privileged Functions

- init

Invariants

- Options can only be minted strictly before `expirationDate`, together with the same amount of Receipts and against the same amount of deposited collateral.
- Exercise is only possible until `exerciseDeadline` (inclusive), and for European options not before `expirationDate`.
- Exercise burns the holder's Options and pays at least `toConsideration(amount, roundUp)` into the Receipt.
- Pair-burn burns equal amounts of Options and Receipts of the same account and returns the collateral 1:1 without a fee.
- Only the holder can `expire` their Options, and only after `exerciseDeadline`.
- A third party can mint, exercise, burn or move an account's Options only with the matching `Perm` bit (or an ERC-20 allowance for moves).
- `init` can only be called once, by the Factory; the template can never be initialized.
- Before `exerciseDeadline`, Receipt supply equals Option supply plus `consBacked`.

Issue_08	Market names omit the window, expiry time and put flag, and `isPut` is an unchecked label
Severity	Low
Description	`optionKey` hashes seven fields, but the name and symbol only show the prefix, the token symbols, the strike and the expiry date. The window, the expiry time of day and a call/put marker are not shown. Because `isPut` inverts the displayed strike without a marker, a market with `(strike, isPut = false)` and one with `(1e36 / strike, isPut = true)` on the same tokens get the same name and symbol, although their economics are opposite: - `OPTA-USDC-WETH-3000-2026-12-25`: the real 3,000 put (USDC/WETH, strike 1/3000, `isPut`), where exercising 1 unit costs 0.000333 WETH per USDC. - `OPTA-USDC-WETH-3000-2026-12-25`: a USDC/WETH call with strike 3000e18, where exercising costs 3,000 WETH per USDC (a worthless long). - `OPTA-WETH-USDC-3000-2026-12-25`: the real 3,000 call (WETH/USDC, strike 3000e18), where exercising costs 3,000 USDC per WETH. - `OPTA-WETH-USDC-3000-2026-12-25`: WETH/USDC with strike 1/3000 and `isPut = true`, where exercising costs 0.000334 USDC per WETH. The last market has the same name as the real 3,000 call and reports `isPut() == true`, yet its strike is close to zero. A writer who picks it by name loses almost all collateral to any long holder. Markets that differ only by their exercise window also share a name. For example, two American WETH/USDC markets with the same expiry but a 1-hour and a 24-hour window have identical names despite different exercise deadlines, so integrators that identify markets by `name()` or `symbol()` can show the wrong deadline or route users to the wrong market.

	The NatSpec tells integrators to read `isPut` instead of the name, but `isPut` is a label the creator sets without any check. Only the token order and the raw `strike` define the economics.
Recommendations	Consider documenting that integrators must identify markets by contract address or `optionKey`, not by name.
Comments / Resolution	Resolved. Names now carry a P/C marker. The window and time of day are still omitted, which is documented in the name() NatSpec together with the advice to identify markets by address or optionKey.

Issue_09	Unsafe keeper example in the `exerciseFor` NatSpec
Severity	**Informational**
Description	The `@dev` of `exerciseFor` contains about 60 lines of example contract code (`FlashExerciseKeeper`), labelled "NOT PRODUCTION READY". Example code that is not meant for production does not belong in the NatSpec of a production contract, where integrators may copy it as a reference.
Recommendations	Consider removing the example from the NatSpec; no code change is needed.
Comments / Resolution	Resolved. The flash-loan example was replaced by a wallet-funded keeper that pays the holder a floor set on-chain, and the spreadWad unit is now clearly defined with an example.

Issue_10	Option and IOption NatSpec states that transfers and pair-burn never stop
Severity	Informational
Description	Several statements describe transfers and pair-burn as permanently available: - `Option.sol:45-47`: "Transfer and pair `burn` never stop … a matched Option+Receipt pair can always be unwound 1:1"; - `Option.sol:322-323`: "Auto-burn stays available for as long as transfers do (forever)"; - `Option.sol:376` and `IOption.sol:250`: `transferFrom` is "likewise open forever"; - `IOption.sol:63-64`: transfer and pair-burn "stay open indefinitely"; - `IOption.sol:324`: pair-burn is "Available at any time". The code stops them: `_settledTransfer` checks `Pause.TRANSFER`, and pair-burn and auto-burn reach `Receipt.burn`, which checks `Pause.BURN`. Neither `IOption` nor its revert lists mention the pause or `ActionPaused`.
Recommendations	Consider correcting the NatSpec statements above; no code change is needed.
Comments / Resolution	Resolved. The Option and IOption NatSpec now describes transfers, auto-burn and pair-burn as not deadline-gated but pausable, and documents the ActionPaused revert and the effect of an EXERCISE pause on the exercise deadline.

Issue_11	Option code quality improvements
Severity	**Informational**
Description	a) `Option.name()` checks `strike() > 0` before inverting a put's strike. This can never be false because the Factory rejects a zero strike, and it costs an extra external call to the Receipt for every put. The `@dev` comment describes a zero-strike case that cannot exist, and `Receipt.name()` already omits the check. b) These `public` functions are never called internally and can be `external`: `init`, `factory`, `mint(uint256)`, `mint(address,uint256)`, `exercise()`, `exercise(uint256)`, `burn(uint256)`, `expire`, `balancesOf` and `details`.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

Issue_12	Option and Receipt names burn most of the caller's gas on WETH markets
Severity	Informational
Description	`Option.name()` / `symbol()` and `Receipt.name()` / `symbol()` render the market name through `OptionUtils.name`, which probes both tokens for a UI multiplier with `supportsInterface` and no gas limit. WETH9 has no `supportsInterface`, so the call reaches its payable fallback, which runs `deposit()` and writes storage. Inside a static call this write fails and consumes all forwarded gas, about 63/64 of the gas left. The `try/catch` then falls back to a 1x multiplier and the name is still returned, but the call costs about 64 times the normal gas (around 979,000 of a 1,000,000 gas limit in testing, against about 21,000 on a market without WETH). Every market with WETH as collateral or consideration is affected, including the production market. Only metadata reads are impacted: callers with a tight gas limit, such as gas-capped RPC calls, multicall batches or contracts with a fixed gas budget, see `name()` and `symbol()` revert. No settlement path reads the name, and no funds are at risk.
Recommendations	Consider capping the gas of the UI-multiplier probe, for example `supportsInterface{gas: 30_000}`, as OpenZeppelin's `ERC165Checker` does.
Comments / Resolution	Acknowledged.

Issue_13	Option and Receipt names revert for tokens whose `name()` or `symbol()` returns `bytes32`
Severity	Informational
Description	`Option.name()` / `symbol()` and `Receipt.name()` / `symbol()` build the market name from the collateral's and consideration's `symbol()`, and `Option.details()` also reads their `name()` and `symbol()`. All of these decode the return value strictly as `string`. Some tokens return `bytes32` instead, such as MKR. Market creation only reads `decimals()`, so a market on such a token is created normally, but decoding a `bytes32` return value as `string` fails. From then on, `name()`, `symbol()` and `details()` of that market's Option and Receipt always revert. Settlement is not affected, but wallets, explorers and integrations cannot display or identify the market.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

Receipt

The Receipt contract is the short side of a market and its vault. It holds all collateral, receives the consideration paid on exercise and tracks `consBacked`, the number of Receipt units now backed by consideration instead of collateral.

Each Receipt is a clone-with-immutable-args of `RECEIPT_CLONE`: the proxy appends 112 bytes of market parameters to the calldata of every call.

Only the paired Option can call `mint`, `burn` and `exercise`.

Appendix: Immutable Arguments

Offset	Field	Size
0	`strike` (18-decimal, consideration per collateral)	32
32	`collateral`	20
52	`consideration`	20
72	`option`	20
92	`expirationDate`	8
100	`exerciseDeadline` = expiration + window	8
108	`isPut`	1
109	`isEuro`	1
110	`decimals`	1
111	`consDecimals`	1

Appendix: Redemption

`redeem()`, `redeem(amount)` and `redeemFor(holders)` (keeper with `REDEEM`) burn Receipts and pay:
1. the **consideration leg** first: `min(amount, consBacked)` units at the strike rate (rounded down), first come, first served, open before expiry on American markets and from `expirationDate` on European markets;

2. the **collateral leg**: the rest, 1:1, only strictly after `exerciseDeadline`.

The holder cannot choose the leg. Pair-burn (through the Option) is not queued and fee-free.

Appendix: Conversion and Fees

`toConsideration(amount)` = `amount * strike * numer / (1e18 * denom)` in one `mulDiv`, where exactly one of `numer` / `denom` is `10^|consDecimals - collDecimals|`. It rounds up when collecting on exercise and down when paying out.

Both redemption legs pay `feeBps` (at most 10%, rounded up), accrued in `feeAccrued` with a 1-wei floor; `collectFees` sends it to the Factory owner. `sweep` (Factory owner) recovers the remaining balance of any token once no Receipt is outstanding.

Privileged Functions

- setFee
- sweep
- collectFees

Invariants

- The collateral balance is at least `totalSupply - consBacked` plus uncollected collateral fees.
- The consideration balance is at least `floor(k * consBacked)` plus uncollected consideration fees.
- `consBacked` never exceeds `totalSupply`.
- The collateral leg only opens strictly after `exerciseDeadline`; European Receipts cannot be redeemed before `expirationDate`.
- Fees are only charged on redemption payouts, and `collectFees` never pays more than the accrued fees.
- `sweep` is only possible when no Receipt is outstanding.
- Only the paired Option can call `mint`, `burn` and `exercise`.
- Conversions always round in favor of the pool.
- With standard tokens and no pause, the last Receipt holder can always redeem in full.

Issue_14	Redeem fee of live markets can be raised to 10% without consent or timelock
Severity	Governance
Description	Each Receipt copies `Factory.feeBps` at creation, but the Factory owner can change it at any time afterwards through `Receipt.setFee`, up to 10% and without a timelock. The new fee applies to every outstanding position of that market. The fee is deducted from both legs of `redeem`, and `collectFees` sends it to `factory.owner()`. A writer who minted at 0% can be repriced to 10% before redeeming. For example, a writer who minted 10 WETH at 0% receives 9 WETH after the fee is raised to 10%, and the 1 WETH fee accrues to the owner. The only fee-free exit for a writer, pair-burn, needs the matching longs and can be paused by the same owner. The repricing power is documented in `IReceipt`, but `Factory.sol:167` and `IFactory` state that the owner "cannot withdraw holder backing", which understates it. The redemption functions take no maximum fee or deadline parameter, so a fee increase also applies to redemptions that are already pending.
Recommendations	Consider fixing the fee of a market at creation, or only allowing decreases on live markets. If increases are required, consider applying them through a timelock with advance notice to users, or adding a maximum-fee parameter to `redeem` so that holders can bound the fee they accept.
Comments / Resolution	Acknowledged.

Issue_15	An outstanding Receipt blocks recovery of unrelated tokens
Severity	**Low**
Description	The Factory owner can use sweep to recover tokens sent to a Receipt contract only when its total Receipt supply is zero. This restriction also applies to unrelated ERC-20 tokens that are neither the market's collateral nor its consideration. For example - There is a sweepable ERC-20 amount in the Receipt contract , another account mints one raw unit of that market's Receipt and transfers it to the Receipt contract itself. The Receipt contract has no way to redeem or transfer the unit it now holds. Its total supply therefore remains nonzero, and the owner cannot sweep the unrelated token, even after the exercise deadline.
Recommendations	Consider acknowledging this issue.
Comments / Resolution	Resolved. sweep now keeps the collateral and consideration backing of outstanding Receipts and allows any other token to be swept at any time.

Issue_16	Receipt and IReceipt NatSpec states that pair-burn and transfers never stop
Severity	Informational
Description	Several statements describe pair-burn and Receipt transfers as permanently available: - `Receipt.sol:40-42` and `IReceipt.sol:24-27`: pair-burn "stays open forever"; - `Receipt.sol:231` and `IReceipt.sol:179-180`: Receipt transfers are "open forever"; - `IReceipt.sol:263-264`: "Writers who want a guaranteed exit should hold (or buy back) the matching longs and pair-burn". The code stops them: `Receipt.burn` checks `Pause.BURN`, as its own `@dev` states, and Receipt transfers check `Pause.TRANSFER`. `IReceipt` does not mention the pause or `ActionPaused`.
Recommendations	Consider correcting the NatSpec statements above; no code change is needed.
Comments / Resolution	Resolved. The Receipt and IReceipt NatSpec now describes pair-burn and transfers as not deadline-gated but pausable (BURN / TRANSFER), documents the ActionPaused revert, and no longer calls pair-burn a guaranteed exit.

Issue_17	Receipt code quality improvements
Severity	Informational
Description	a) These `public` functions are never called internally and can be `external`: `mint`, `burn`, `exercise`, `redeem()`, `redeem(uint256)` and `toCollateral`. b) The fee cap `1000` (`Receipt.sol:217`) and the basis-point denominator `10_000` (`Receipt.sol:358`) are literals. The cap must not be lower than the Factory's cap, because every new Receipt is initialized with the Factory's fee.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

Issue_18	A zero net payout reverts for tokens that reject zero-value transfers
Severity	**Informational**
Description	`_payout` only skips the transfer when the gross amount is zero. With a non-zero fee, a gross amount of one raw unit becomes a fee of one unit, so `paid == 0`, and `safeTransfer(account, 0)` is still executed. The consideration leg is always paid first. When the conversion rate is between 1 and 2 raw units per Receipt (for example an 18/18-decimal pegged pair at strike 1.0), a `consBacked` residue of one unit, which anyone can create with a one-unit exercise, makes every `redeem` and `redeemFor` pay one unit gross and zero net. With a consideration token that reverts on zero-value transfers (such as BNB on Ethereum), every redemption of the market reverts while `consBacked == 1`. Before the deadline a further exercise can lift it, but after the deadline only the owner can unblock the market by setting its fee to 0. Pair-burn still works.
Recommendations	Consider skipping the token transfer when the net payout is zero.
Comments / Resolution	Acknowledged.

Issue_19	Sweeping one address of a dual-address token leaves the other address's fee tally unbacked
Severity	Informational
Description	`Receipt.sweep(token)` resets `feeAccrued` only for the token address it is called with, but transfers that address's whole balance. If a market uses a token with two addresses over one balance ledger (a double-entry-point token), for example as collateral and consideration, or as a consideration that also has a second live address, sweeping through one address also drains the balance that backs the other address's `feeAccrued`, while that tally stays unchanged. On an American market swept before expiry and then minted into again, the permissionless `collectFees` pays the stale tally to the Factory owner out of the new writers' collateral, so later pair-burns and redemptions revert for lack of funds.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged. Tokens with two addresses over one balance ledger are documented as unsupported in the sweep NatSpec.

Permissions

Permissions defines the bit flags used by the Factory registry and pause mask. It contains constants only, but its NatSpec is the specification of the permission registry.

Privileged Functions

- None.

Invariants

- Only bits inside `Perm.ALL` / `Pause.ALL` can be set.
- `TRANSFER`, `TRANSFER_RECEIPT`, `REDEEM` and `EXERCISE` have no effect on the account's own row; `MINT` and `BURN` on the own row are opt-ins for transfer-time automation.
- `BURN` cannot reach `expire`.
- A pause bit only blocks the actions listed for it.

No issues were identified in this contract.

ClonesWithImmutableArgs / Clone

`ClonesWithImmutableArgs` builds the creation code of the Receipt proxies and deploys them with CREATE (`clone`) or CREATE2 (`clone2`, with a caller-supplied salt). The init code is a 10-byte prelude that returns the runtime: 55 bytes of forwarder (copy calldata, append the immutable args from the proxy's own code, DELEGATECALL the implementation, bubble the result), followed by the args and a `uint16` length suffix.

`MAX_DATA_LENGTH = 24519` keeps the runtime within the EIP-170 limit.

`Clone` provides the `_getArg*` readers used by the Receipt, which locate the args from the last two bytes of calldata.

Both contracts are `internal`-only:
`ClonesWithImmutableArgs` is inlined into the Factory and `Clone` is inherited by the Receipt, so nothing is linked.

The bytecode and the reader offsets were verified byte for byte against the upstream implementation.

Privileged Functions

- None.

Invariants

- `creation()` returns the prelude, the 55-byte runtime, the data and `uint16(data.length)`.
- The runtime never exceeds the EIP-170 limit; longer data reverts with `DataTooLong`.
- `_getImmutableArgsOffset` always locates the appended args, whatever extra calldata a caller adds.
- All library functions are internal, so no library is linked or delegatecalled.

Issue_20	Clone library code quality improvements
Severity	**Informational**
Description	a) `Clone.sol` and `ClonesWithImmutableArgs.sol` declare `pragma solidity ^0.8.4`, but `ClonesWithImmutableArgs.sol` uses `assembly ("memory-safe")`, which requires solc 0.8.13 or later. The declared range is therefore wrong. The in-scope contracts use `^0.8.30` and the project compiles with 0.8.35. b) `ClonesWithImmutableArgs.sol:76-77` names `Factory.receiptInitCodeHash` as an example consumer, but the Factory exposes no init-code-hash view. c) `Clone._getArgUint256Array` is never called: the Receipt only uses `_getArgAddress`, `_getArgUint256`, `_getArgUint64` and `_getArgUint8`. d) `ClonesWithImmutableArgs.sol:55-70` keeps a commented-out `addressOfClone2` function in the NatSpec, while `lib/clone/README.md` presents it as part of the library API.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Resolved. foundry.lock now pins lib/clone to greekfi/clone 13741b2.

Issue_21	Receipt clones can dispatch calls shorter than 4 bytes on a selector formed from `strike`
Severity	Informational
Description	Each Receipt is a clone-with-immutable-args proxy that appends 112 bytes of market arguments, starting with `strike`, to the calldata of every call before delegating to the Receipt implementation. Because the appended data makes the calldata longer than 4 bytes, the Solidity dispatcher always reads a selector, and for a call shorter than 4 bytes that selector is formed from the caller's bytes followed by the leading bytes of `strike`. With any realistic strike these leading bytes are zero. The Receipt has no fallback and no selector ending in `0x00`, so the call reverts. Only a market with a strike above roughly 2^245, where exercise is economically impossible, could dispatch a short call to a real function, and an impact would additionally need a contract that holds that market's Receipts and forwards untrusted short calldata to it.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

4. Bebop Integration

The integration was reviewed against the frozen Greek commit and the deployed Bebop settlement contract on Ethereum, in the configuration confirmed by the Bebop and Greek teams and observed in production.

4.1 Scope

The integration review is based on the client's end-to-end test https://github.com/greekfi/contracts/blob/main/test/BebopE2E.t.sol, reviewed at commit `c7e2027c87a22c851062a94073ece0e8d0af4b88`. It covers the flow that test exercises: a writer selling Greek Options through Bebop with auto-mint.

In scope:

- the Greek contracts at the audited commit, as far as the flow uses them: `Option.transferFrom` and its auto-mint and auto-burn legs, the Factory permission registry and `Factory.transferFrom`, and the `Perm` bits;
- the whole BebopSettlement contract `0xbbbbbBB520d69a9775E85b458C58c648259FAD5F` on Ethereum (verified source). `swapSingle` with a standard token transfer (`packed_commands = 0`) is the planned entry point; the other entry points and order options were reviewed for how they interact with the Greek permission model, and the ways they can cause harm when used incorrectly are reported as issues below;
- the production configuration confirmed by the Bebop and Greek teams and observed on Ethereum in the reference fill.

Out of scope:

- all other Bebop contracts (BebopRouter, JamSettlement, the JAM Balance Manager) and any route that settles through them, as well as the internals of Permit2;
- Bebop's off-chain systems (RFQ API, quote service, market listing and its filters, maker selection) and the Greek frontend;
- the deployed Greek Factories, whose code differs from the audited commit, and deployments on chains other than Ethereum.

Appendix: Contracts Checked on Ethereum

Contract	Address	Notes
BebopSettlement	`0xbbbbbBB520d69a9775E85b458C58c648259FAD5F`	Settles the orders and calls `Option.transferFrom`. Verified source (Blockscout, Sourcify "match"), solc 0.8.19, created on 2024-05-02 (block 19,783,283). No owner, proxy, pause or upgrade path.
Example market	Option `0xFcA3B7F77C62Ccd05406347Fefc2B3297f5044E2` Receipt `0xF12D0B237237d6c516a7Ec3F1eEAC1a77dcB599d`	`OPTE-WETH-USDC-2750-2026-10-09`: European WETH/USDC call, strike 2,750, 1-hour exercise window, redeem fee 0.
Reference fill	tx `0x485b4402a000fbfce6facacca5969fa3dd1c3472800aee9c7340eafb37f13ea3`	A writer sells 0.01 Option for 0.141525 USDC through `swapSingle`, with auto-mint.

Bebop's other contracts (BebopRouter `0xBeb0009ACa35087ce7cCF11637E24dd1Aad3bf2A`, JamSettlement `0xbeb0b0623f66bE8cE162EbDfA2ec543A522F4ea6` and the JAM Balance Manager `0xC5a350853E4e36b73EB0C24aaA4b8816C9A3579a`) are not used by the integration and were not validated for it. Only BebopSettlement was reviewed, as described in 4.1.

4.2 Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)	Failed resolution	Open
High						
Medium						
Low	1			1		
Informational	4			4		
Governance						
Total	5			5		

4.3 Detection

Greek Options are sold through Bebop's RFQ (request-for-quote) settlement without minting them first. The Options are minted just in time (JIT) by the Option's auto-mint: when a transfer exceeds the sender's balance and the caller holds `MINT` on the sender's permission row, the Option mints the missing amount against the sender's collateral inside that same transfer. The writer approves the collateral to the Greek Factory and grants the Bebop settlement contract Factory permission bits on the writer's permission row. A Bebop market maker signs an order that buys the Option for a premium. The writer submits it, and the settlement calls `Option.transferFrom` from the writer to the maker. Because the writer holds no Options yet, auto-mint pulls the collateral through the Factory and mints the missing Options and Receipts to the writer inside the same transaction. The writer ends with the Receipts (the short position) and the premium, and the maker ends with the Options.

Appendix: Order Flow

1. Listing: Greek creates markets through the Factory, and Bebop lists them by indexing the `OptionCreated` events, filtered by the market creator and the tokens.
2. One-time grant: the writer calls `factory.addPermissions(BebopSettlement, mask)`. The reference test grants `TRANSFER | MINT`; production writers grant `TRANSFER | MINT | BURN`.
3. Per trade: the writer approves the collateral (e.g. WETH) for the sold amount to the Factory.
4. Quote: a Bebop maker signs an EIP-712 `SingleOrder` with `taker_address = receiver = writer`, `taker_token = Option`, `maker_token = USDC`, `taker_amount` = Options sold, `maker_amount` = premium, `packed_commands = 0`, partner id 0 and a short expiry (15 seconds in the reference fill).
5. Submission: the writer calls `swapSingle(order, makerSignature, 0)` on BebopSettlement, which reverts unless `msg.sender == order.taker_address`.
6. Validation: BebopSettlement verifies the maker's signature, consumes the maker's nonce and checks the expiry.
7. Taker leg: BebopSettlement calls `Option.transferFrom(writer, maker, amount)`. `TRANSFER` in the writer's row lets it skip the Option allowance. The writer's Option balance is short and `MINT` is granted, so the Option mints the shortfall: the Receipt pulls the collateral from the writer through `Factory.transferFrom`, and the Option and Receipt are minted to the writer. The Options then move to the maker.
8. Maker leg: BebopSettlement transfers the premium from the maker to the receiver.

Appendix: Auto-Mint Conditions

An auto-mint fill only succeeds when all of the following hold:

- the contract that calls `Option.transferFrom` is the one the writer granted `MINT`; only direct BebopSettlement fills with a standard transfer satisfy this, while Permit2, BebopRouter and JAM routes call from other contracts and revert.
- `block.timestamp < expirationDate`.
- `Pause.TRANSFER` and `Pause.MINT` are not set.
- the writer holds the collateral and has approved it to the Factory.

Only the shortfall over the writer's existing Option balance is minted.

Appendix: Access Control on the Bebop Side

- `swapSingle` and the other `swap*` functions only accept the order's taker as `msg.sender`.
- `settle*` functions accept any submitter, but require the taker's own signature. Takers cannot use delegated signers.
- Every order requires the maker's signature over the partner id, expiry, taker, maker, nonce, both tokens, both amounts, the receiver and `packed_commands`. The maker nonce can only be used once.

The settlement can therefore only move a writer's Options in a transaction the writer sends or an order the writer signs.

Appendix: Checks for Makers and Frontends

Market names and the `isPut` flag do not define a market's economics. Before quoting or displaying a market, read the collateral, consideration, raw strike, `expirationDate`, `exerciseDeadline` and `isEuro` from the Receipt. Derive call or put from the token order, size amounts in collateral units, and check the market's current `feeBps`.

Appendix: Assumptions

The following were confirmed by the Bebop and Greek teams and are treated as trusted:

- production orders are settled only by BebopSettlement `0xbbbb...FAD5F`.
- makers are Bebop's integrated partner market makers.
- Bebop lists only markets that pass its creator and token filters.

Privileged Functions

- Writer: Grants and revokes the settlement's permission bits, and controls the collateral approval to the Factory.
- BebopSettlement: None.

Invariants

- BebopSettlement moves a writer's Options only in a `swap*` sent by the writer or a `settle*` carrying the writer's own signature.
- A fill is atomic: the auto-mint, the delivery of the Options and the premium payment happen together or not at all.
- Auto-mint mints only the writer's shortfall, 1:1 against collateral pulled from the writer through the Factory, and only before `expirationDate`.
- Every Option sold either comes from the writer's balance or is minted together with one Receipt for the writer.
- The maker's signature fixes the tokens, amounts, receiver, partner id and `packed_commands`, and each maker nonce can be used once.
- The writer's grant only takes effect when BebopSettlement itself calls `Option.transferFrom`.

Issue_22	A `BURN` grant to the Bebop settlement lets third parties pair-burn a writer's Receipts
Severity	Informational
Description	When an Option transfer is made by an operator that the receiver granted `BURN`, the auto-burn leg pair-burns the incoming Options against the receiver's Receipts and returns the collateral 1:1. This is intended for buy-back netting: a writer who buys back its Options through Bebop has its short closed in the same transaction. A normal transfer from another account does not trigger the auto-burn, because that account holds no `BURN` grant from the receiver. BebopSettlement, however, delivers Options to `order.receiver`, a field chosen by the parties of the order, not by the receiver. If a writer granted `BURN` to BebopSettlement, any account can therefore send it Options through Bebop and pair-burn its Receipts, closing its short at a moment the writer did not choose. The grant is not tied to a market, so this applies to the writer's Receipts in every market of the Factory. The writer always receives the collateral at par, so no funds are lost; the only effect is that a third party decides when the writer's position is closed. Selling through Bebop only needs `TRANSFER | MINT`, and `Permissions.sol` advises granting `BURN` only to contracts that do not let third parties choose the recipient of an Option transfer.
Recommendations	This issue is informational. Consider acknowledging it.
Comments / Resolution	Acknowledged.

Issue_23	Post-signing registration of a Bebop partner ID can redirect all maker output
Severity	Low
Description	Bebop signatures commit to the numeric `partnerId`, but not to its registration status, fee or beneficiary. An unregistered ID applies no fee, yet anyone can later register it with `registerPartner` with a fee of up to 10,000 basis points and an arbitrary beneficiary. Registration does not invalidate existing signatures. Settlement reads the partner configuration at execution and deducts the fee from the maker's output. Therefore, an order signed while its partner ID is unregistered can execute under materially different terms after registration. For example, a writer agrees to sell one Option for 100 USDC using an unregistered partner ID. Before execution, the maker registers that ID with a 100% fee and itself as beneficiary. When the writer executes: • the Option is transferred or auto-minted to the maker; • the maker's 100 USDC output is treated entirely as a partner fee and returned to the maker; • the writer receives zero USDC and is left holding only the Receipt, with its collateral locked and the resulting short exposure. An unrelated account could similarly register the ID and receive the maker output itself. Partner ID 0 is pre-registered with no fee and cannot be registered again, and a registered partner's fee and beneficiary can never change. Only non-zero partner IDs that are unregistered at signing are affected. The root cause is in Bebop's settlement contract, so it cannot be prevented in the Greek contracts.
Recommendations	Consider accepting only orders with partner ID 0 in the Greek frontend

	and the Bebop quote service.
Comments / Resolution	Acknowledged.

Issue_24	The writer's grant to the Bebop settlement is broader than the sale requires
Severity	**Informational**
Description	The writer grants BebopSettlement `TRANSFER | MINT` on its Factory permission row. The grant applies to every market the Factory has created or will create, stays active after the fill, and is not an ERC-20 allowance, so wallets and allowance dashboards do not show it. `TRANSFER` is not needed for the sale. Without it, `Option.transferFrom` spends the ordinary Option allowance, and the auto-mint leg only checks `MINT`. The same fill therefore works with `MINT` plus `option.approve(BebopSettlement, amount)`, which limits the settlement to the chosen market and amount. With `TRANSFER`, any order the writer submits or signs on BebopSettlement can move the writer's longs in any market, and `MINT` can mint new ones up to the writer's collateral approval to the Factory.
Recommendations	Consider granting BebopSettlement only `MINT` and approving the specific Option for the sold amount before each fill. Also consider showing the active grants in the frontend and revoking them with `setPermissions(BebopSettlement, 0)` when the writer stops trading.
Comments / Resolution	Acknowledged.

Issue_25	Public taker authorizations allow partial fills to invalidate entire RFQ-M orders
Severity	Informational
Description	On the `settle*` entry points of BebopSettlement, anyone holding the maker and taker signatures can submit the order. Neither signature binds `filledTakerAmount`, and every successful fill, of any size, consumes the maker's complete nonce without recording the remaining fillable amount. When a valid taker authorization is exposed through a pending `settle*` transaction, another party can reuse it with a smaller positive fill. If the modified settlement succeeds first, the maker nonce is consumed and the intended settlement reverts. Depending on the order's raw amount ratio, rounding down may cause a dust fill to transfer zero or negligible maker output. With Greek Options, the writer is also left with a dust Option and Receipt pair minted by the auto-mint. The affected parties must replace the consumed order and obtain new signatures. This causes gas loss, execution delay, loss of the quoted price or a temporary hedge mismatch, and the griefing can be repeated whenever a replacement authorization is exposed again. The attacker gains nothing and no funds are lost. The planned route, `swapSingle`, is not affected, because only the taker can submit it and choose the fill size. The root cause is in Bebop's settlement contract, so it cannot be fixed in the Greek contracts.
Recommendations	Consider documenting that Greek fills should be executed through `swapSingle` by the writer, where only the taker can submit and choose the fill size. If the `settle*` route is ever used, consider submitting orders through a private relay so that taker signatures are not exposed in the public mempool before inclusion.
Comments / Resolution	Acknowledged.

Issue_26	Per-entry Bebop rounding can erase duplicated low-decimal Option amounts
Severity	Informational
Description	Greek Options inherit the collateral token's decimals. A GUSD-collateral Option therefore has two decimals, making one raw Option unit equal to 0.01. During partial fills, BebopSettlement scales and rounds down every Multi or Aggregate array entry independently, and it allows the same token address to appear in several entries. A signed Multi order can split `1.00` Option into 100 outputs of `0.01`. Under a near-full partial fill, each entry becomes `floor(1 × filledAmount / fullAmount) = 0`, so the taker pays almost the full signed amount while receiving zero Options. The Aggregate variant applies the same calculation to signed `TRANSFER_FROM_CONTRACT` inputs. If a maker signs 100 one-atom Option inputs, an Aggregate executor can apply a near-full global fill ratio that reduces every input to zero while the maker still transfers almost all of its output. The executor cannot split a normal order after signing: the repeated entries must already be present in the signed arrays. The planned route, `swapSingle`, has a single entry per leg and is not affected. The root cause is in Bebop's settlement contract, so it cannot be fixed in the Greek contracts.
Recommendations	Consider documenting that Greek fills should use `swapSingle`, and having the Greek frontend and the Bebop quote service reject Multi or Aggregate orders that repeat the same Option in several entries or contain entries of only a few raw units.
Comments / Resolution	Acknowledged.