

Report

Security Report

by Asymptotic

Overview

This report covers only Blast's launchpad, the `blast_fun` package, which lets creators launch a new coin (the Meme) on a constant-product bonding curve priced in an admitted Quote coin such as SUI. Each launch is a shared `Launch` object that holds the curve's reserves, the fees it earns and the allocations it owes until its real Quote reserve reaches the creator's graduation target. An admitted migration adapter then calls `migrate`, which pays the launch's recipients, disposes of the Meme the curve did not sell and hands the pool's share of both coins to the adapter for a DEX pool.

Key features

- **Generated coins with a fixed supply.** Each Meme comes from its own generated package, registered in Sui's Coin Registry with exactly one billion whole tokens minted into a `LaunchTicket`. Finalizing the launch fixes the supply as burn-only or fixed.
- **Virtual-reserve bonding curve.** Trades price against the Meme reserve and the creator's virtual Quote reserve plus the real Quote reserve, with a Meme fee and a Quote fee on every trade. A creator-selected share of each sell's Meme is held back and burned at migration.
- **Optional opening round.** For one to five minutes after the launch is shared, bidders pool Quote. Settlement buys from the curve once with the whole pot, and each bidder receives a pro-rata share of the Meme bought and of any Quote refunded.
- **Per-launch fee accounting.** Each launch keeps its own creator and Blast fee balances, plus the SUI creation fee. The creator claims with a revocable `CurveFeeCap`; Blast claims with its treasury role and pays a named partner its share in the same call.
- **Creator authority.** The creator recipient, either an address or a Blast account attested by a backend signature, claims a curve-fee cap and a metadata cap once. An administrator can reassign either.

Notable specifics

- **Typed administrator roles.** Every administrative entry point accepts the witness of exactly one role (upgrades, fees, treasury, listings,

adapters, creator caps or the claim signer), issued by the shared Blast ACL in `blast_admin`.

- **Policy snapshotted per launch.** Trade fees and Blast's allocation are copied into each launch when it is created, and the partner a launch names never changes, so later configuration changes reach only new launches. Blast's share of each trade fee is read live from `RuntimeConfig`.
- **Hard bounds on rates.** Each trade fee is capped at 10% and the sell burn at 50%. Blast's allocation always leaves a share for the curve and the pool, and a partner earns at most 80% of Blast's fee.
- **Custody stays in the launch.** Reserves and fees are `Balance` fields of the shared `Launch`. The coin's native metadata capability never leaves it.
- **Regulated Quotes.** Regulated Quote coins are admitted by design, and the client accepts that an issuer's deny list or pause can block payouts in that coin, including migration ([ADR 0088](#)). Protocol-fee claims pay Meme, Quote and SUI in one transaction, so a deny-listed treasury also blocks the SUI and Meme fees and the partner's share. A deny-listed treasury can be re-pointed with `set_treasury`.
- **Generation-gated upgrades.** A shared `PackageVersion` records the active generation, and gated entry points reject superseded bytecode. Upgrades go through a `PackageAdmin` that seals the `Publisher` and `UpgradeCap` and authorizes and commits in one transaction. The commit advances the generation from the outgoing bytecode, so a release that does not bump `PACKAGE_VERSION` locks every gated entry point, including further upgrades.
- **Not yet published.** The package is still pre-publication (`0x0` in `Move.toml`). The remediation changed eleven public signatures, so the final reviewed commit must be published as a new package, not as an upgrade of the review start commit.

Our results are summarized in [Summary of Results](#).

TABLE OF CONTENTS

Scope

Summary of Results

Findings

Launchpad goes live with the UpgradeCap address-owned; the required PackageAdmin wrap is unenforced

A sub-atom gap below the graduation target makes every buy revert until someone sells

Undelayed launchpad upgrades rest entirely on undocumented UpgradeAdmin custody

Undocumented root SuperAdminCap custody gives its holder undelayed upgrade control of the launchpad

Launch finalization and the LaunchInitializer setters skip the package-generation guard

fees::new accepts Meme == Quote, leaving FeeLedger's four-bucket separation to the caller

errors_tests registry pins only 35 of the 38 error macros

Stale QuoteSet doc describes a removed virtual-reserve field

ProtocolAllocationSet doc implies the new share applies to existing launches

metadata_cap_id names two different cap types across launchpad events

Formal Verification

Trusted Dependencies

Coverage and Limitations

Methodology

Commits

Scope

Project: blast-v2-contracts

Reviewed: only the in-scope packages below, not the rest of the repository

Branch: `commit-launchpad`

Review start commit: `b549bb16e02e4c6e8ff59e1958435a638c784973`

Final reviewed commit: `01d7659617abcd552cece6d05b9db409ec4ff2c0`

IN-SCOPE PACKAGE	PATH
<code>blast_fun</code>	<code>sui/launchpad</code>

- **In scope**
 - `blast_fun` (`sui/launchpad`): the launchpad, covering the launch lifecycle and bonding curve, fee custody and claims, creator authority, launch and runtime configuration, package administration and the migration handoff.
- **Out of scope**
 - The rest of the Blast v2 repository: presale, Blast SUI, boosts, migrators and the generated-coin template. This report makes no statement about them.
 - `blast_admin`, `blast_partners` and `blast_fun_timelock`. The launchpad depends on them, and we treat them as trusted.
 - Migration adapters and the DEX venues that receive a launch's liquidity. The launchpad checks only the adapter's admitted witness type before it hands over the pool's share.
 - Off-chain services, including the backend that signs account claims.
- **Remediation review:** the fixes for the findings below were reviewed at the final reviewed commit `01d76596`. The client's fixes were merged in `803ebfc6`.
- **Dependencies:** the packages above, the Sui Framework and the Move Standard Library. [Trusted Dependencies](#) sets out what each is trusted for.

Summary of Results

SEVERITY	CONFIRMED FINDINGS
CRITICAL	0
HIGH	0
MEDIUM	0
LOW	5
INFORMATIONAL	5

Our review of the launchpad (`blast_fun`) finds no critical, high or medium severity issues, and no finding lets an untrusted party take or redirect funds. The low-severity findings concern deployment and administration hardening (sealing upgrade authority in `PackageAdmin` before launch, documenting who holds the root and upgrade keys, and applying the package-generation guard to every launch-mutating function) and a narrow window near the graduation target in which buys revert until a Meme holder clears it. The informational findings cover a missing defensive check, an incomplete error-code test, and event documentation and naming. The repository's other packages are outside this review.

Remediation. The client remediated L-1, L-2, L-5 and all five informational findings, and acknowledged L-3 and L-4, which they will address with a multisig for the root and upgrade roles. We reviewed each fix at the final reviewed commit `01d76596` and confirmed that it resolves its finding.

Findings

ID	TITLE	SEVERITY	STATUS
L-1	Launchpad goes live with the UpgradeCap address-owned; the required PackageAdmin wrap is unenforced	LOW	Remediated

ID	TITLE	SEVERITY	STATUS
L-2	A sub-atom gap below the graduation target makes every buy revert until someone sells	LOW	Remediated
L-3	Undelayed launchpad upgrades rest entirely on undocumented UpgradeAdmin custody	LOW	Acknowledged
L-4	Undocumented root SuperAdminCap custody gives its holder undelayed upgrade control of the launchpad	LOW	Acknowledged
L-5	Launch finalization and the LaunchInitializer setters skip the package-generation guard	LOW	Remediated
I-1	fees::new accepts Meme == Quote, leaving FeeLedger's four-bucket separation to the caller	INFORMATIONAL	Remediated
I-2	errors_tests registry pins only 35 of the 38 error macros	INFORMATIONAL	Remediated
I-3	Stale QuoteSet doc describes a removed virtual-reserve field	INFORMATIONAL	Remediated
I-4	ProtocolAllocationSet doc implies the new share applies to existing launches	INFORMATIONAL	Remediated
I-5	metadata_cap_id names two different cap types across launchpad events	INFORMATIONAL	Remediated

Launchpad goes live with the UpgradeCap address-owned; the required PackageAdmin wrap is unenforced

L-1

LOW

Remediated

Location: `blast_fun`

Description. At publish, `blast_fun::init` (`blast_fun.move:32-35`) shares a `PackageVersion` at generation 1 and sends the package's `Publisher` to the deployer, while the Sui framework delivers the raw `UpgradeCap` to the same address. Moving both into the shared `PackageAdmin` takes a separate, optional transaction, `package_admin::wrap` (`package_admin.move:20-25`), and no entry point requires it. The launchpad's generation check already passes at publish, so every `launch_config`, `launch` and `runtime_config` entry point works normally whether or not `wrap` has run.

ADR 0031 and ADR 0032 describe the wrap as required and state that both capabilities must leave the deployer key, but nothing in code enforces the order. The gap is also one-way, because `upgrade_custody::new` accepts only an `UpgradeCap` for the original package at native version 1 (`upgrade_custody.move:37-42`). If the deployer upgrades with the raw cap before calling `wrap`:

- the cap's version advances, and custody for this package lineage can never be established;
- the generation coupling is lost as well: `commit_upgrade_and_advance`, the only call that advances `PackageVersion` and so stops superseded bytecode, is reachable only through a `PackageAdmin`.

The proven specification of `wrap` states exactly this abort condition (see [Formal Verification](#)). Sibling packages ship a deployment runbook that orders these steps, such as the presale package's [DEPLOYING.md](#); the launchpad, which holds user funds, has none.

Impact. A missed or misordered deployment step leaves upgrade authority with a single deployer key. It can also permanently prevent moving that authority under role-gated custody, together with the generation cutover that retires old bytecode. Only the deployer, who holds both capabilities after publish, can cause this; no third party can force, block or front-run it. The package is not yet published (`blast_fun = "0x0"` in [Move.toml:12](#)), so this is an unenforced deployment requirement rather than a live exploit.

Recommendation. Before the first publish, enforce the order between publish and `wrap`:

- ship a `sui/launchpad/DEPLOYING.md` runbook, mirroring the presale one, that requires calling `wrap` immediately after publish and forbids any raw upgrade before it; or
- hold the `Publisher`, the `UpgradeCap` and the initial generation in an owned bootstrap object that only `wrap` can consume, so no gated entry point is reachable until custody is sealed.

Remediation note. Until `wrap`, the raw `UpgradeCap` is still in the deployer's wallet. A raw upgrade or policy change in that window fails closed (the launchpad never activates) rather than going live without custody. The deployer must wrap right after publishing, as `DEPLOYING.md` requires.

A sub-atom gap below the graduation target makes every buy revert until someone sells

L-2

LOW

Remediated

Location: `blast_fun::launch`

Description. Near graduation, `launch::buy_from_curve` (`launch.move:772-824`) uses `curve::cap_buy_quote_input` (`curve.move:14-32`) to cap each buy's gross input at exactly what reaches `graduation_target`. When the remaining gap is worth less than one Meme atom at the current curve price, the capped input buys nothing:

- `curve::quote_buy` rounds the Meme output down to zero;
- `buy_from_curve` returns the input untouched before it joins any reserve (`launch.move:793`);
- `buy` then aborts with `zero_trade_output` (`launch.move:532`).

Because the cap, not the buyer's input size, fixes the output, every buy of any size aborts the same way while the reserve sits in this window. Until it leaves the window, the launch cannot move from `Live` to `MigrationReady` through a buy, so graduation stalls.

`finalize` checks only that `virtual_quote` is positive and that the pool's share of the target is positive (`launch.move:216` and `launch.move:230-231`). Nothing bounds `virtual_quote` and `graduation_target` against the Meme's decimals, so the creator's parameters set the width of the window,

which is small: around one Quote atom in a SUI-denominated example. Both values are public, so any address can size an ordinary buy to land the reserve inside the window and stall the launch on purpose, at the cost of its own trade fees.

The stall clears in one transaction. Any Meme holder, including whoever triggered it, can `sell` (still open while `Live`) to widen the gap and then `buy`. That buy lands exactly on the target and moves the launch to `MigrationReady`. This contradicts [ADR 0043](#), which expects a final buy sized to reach the target to always succeed.

Impact. Griefing only. No funds are at risk and nothing moves to the attacker, but buys revert and graduation waits until someone performs the sell-then-buy round trip, paying one extra round of trading fees.

Recommendation. Make the terminal step independent of a positive Meme output, with either or both of these compatible changes:

1. In `buy_from_curve`, when the capped input brings `real_quote_reserve` to the target, apply it and move the launch to `MigrationReady` even when the Meme output rounds to zero, and let `buy` accept that terminal case instead of reverting.
2. At `finalize`, bound `virtual_quote` and `graduation_target` against the admitted Meme decimals, so the price of one atom near the target stays below a known-safe bound.

Cover the change with a boundary test that buys to exactly one atom's price below the target and confirms that the launch reaches `MigrationReady` without an intervening `sell`.

Remediation note. When one Meme atom costs more than the remaining gap, the buy that graduates the launch can pay that gap and receive 0 Meme. Buyers can opt out with `min_meme_out ≥ 1`. Opening-round settlement always passes 0, so bidders can't. The amount involved is less than the price of one Meme atom. Indexers should accept `Bought` events with `meme_out_net = 0`.

Undelayed launchpad upgrades rest entirely on undocumented UpgradeAdmin custody

L-3

LOW

Acknowledged

Location: `blast_fun::package_admin`

Description. `package_admin::authorize_upgrade` and `commit_upgrade_and_advance` ([package_admin.move:28-52](#)) let any `AdminCap` holding the `UpgradeAdmin` role authorize and commit an arbitrary package digest in one transaction. Neither call records a pending digest or reads the clock. The upgrade policy stays `compatible` for good: the wrapper offers no way to narrow it, only `make_immutable` ([package_admin.move:55-64](#)).

The role itself is granted without delay. `blast_acl::grant_role` ([blast_acl.move:89-104](#)) needs only the `SuperAdminCap` and applies at once; the ACL's 7-day timelock covers only replacing the root. The proven specifications of both calls list every abort condition (the current generation, the sealed cap, a 32-byte digest or a matching receipt, and the in-flight state), and none of them depends on time.

Under the compatible policy, new bytecode can rewrite the logic that governs every launch's curve reserves, the creator and Blast fee balances, and partner payouts. `PackageVersion.advance` then retires the old code, so users get no window to exit. [ADR 0084](#) records that an on-chain upgrade delay was proposed for a sibling package and declined. The package therefore relies entirely on off-chain custody of the root and of the `UpgradeAdmin` caps. The launchpad's documentation states no custody requirement, and [ADR 0007](#) assigns upgrade authorization to the "protocol operations" `AdminCap` rather than the multisig.

Impact. A compromised or malicious holder of an `UpgradeAdmin` cap could take every launch's reserves and all accrued fees. No untrusted user can trigger this, so it is a centralization risk rather than an exploit.

Recommendation.

- Hold the `SuperAdminCap` and every `AdminCap` granted `blast_fun_domain::UpgradeAdmin` in a disclosed multisig, and document this in the launchpad's deployment documentation, as the presale package already does.
- If users need time to react to code changes, add a pending-digest delay modelled on the ACL's root-transfer timelock, or commit publicly to an off-chain upgrade notice period.
- Call `make_immutable` once the package is stable.

Client response. Acknowledged. The client accepts the risk for now. Once the multisig is set up, they will document its address, signers, threshold and

ownership of the root and upgrade roles, then verify custody. Upgrades will remain immediate under multisig control.

Undocumented root SuperAdminCap custody gives its holder undelayed upgrade control of the launchpad

L-4

LOW

Acknowledged

Location: `blast_admin::blast_acl`

Description. The ACL lives in `blast_admin`, a trusted dependency outside this review; this finding concerns what custody of its root means for the launchpad. `blast_acl::init` ([blast_acl.move:49-54](#)) sends the only `SuperAdminCap` to whichever address publishes the immutable ACL package. The cap is `key`-only, so it stays at that address until a 7-day `begin_super_admin_transfer` and `accept_super_admin_transfer` completes. By design ([ADR 0097](#), [ADR 0015](#)), the root's `grant_admin` and `grant_role` take effect immediately. The holder can therefore make itself `blast_fun_domain::UpgradeAdmin` and run `package_admin::authorize_upgrade` and `commit_upgrade_and_advance` within two transactions, with no delay. The same applies to the launchpad's `TreasuryAdmin` and its other roles.

Upgrade authority over the launchpad protects every curve reserve and fee balance, yet no accepted document states how the root will be held:

- [ADR 0007](#) ("protocol multisig") is only proposed;
- [ADR 0053](#) says a maintainer multisig "may" own it;
- [ADR 0015](#) and [ADR 0061](#) call it a "cold root" but name no holder type, signer set or threshold;
- [README.md](#) requires recording the holder's address, but not whether it is a multisig or its threshold.

Impact. The risk is operational. If the publisher is a single key, losing or compromising that key gives immediate, undelayed upgrade control over the launchpad, and through it over all curve reserves and fee balances. No untrusted actor can trigger anything here, and nothing is exposed yet because the `blast_admin` lineage is not published.

Recommendation. State the intended root custody (a multisig, its signer set and threshold) in an accepted decision record or runbook, and publish `blast_admin` from that address. Record the custody form and threshold

alongside the ACL and `SuperAdminCap` IDs in the deployment record. The launchpad deployment runbook recommended in L-1 is the natural place to record it.

Client response. Acknowledged, together with L-3. Since the reviewed commit, `blast_acl::init` delivers the root to a configured `blast_acl_root` address instead of the publisher (`a9da80f1`), currently the deployer of the published `blast_admin` package (`acad02ec`). The client will move the root to the multisig and document its address, signers and threshold once it is set up.

Launch finalization and the LaunchInitializer setters skip the package-generation guard

L-5

LOW

Remediated

Location: `blast_fun`

Description. Every function in `launch.move` that mutates a shared `Launch` takes a `&PackageVersion` and calls `assert_current()`. The creation path is the exception; these functions take none:

- `finalize` (`launch.move:188-288`) and `share` (`launch.move:292-301`);
- the `LaunchInitializer` setters `set_virtual_quote`, `set_graduation_target`, `set_allocation_plan`, `set_lp_fee_split`, `set_opening_round` and `set_partner` (`launch.move:131-182`).

This breaks the module's own documented invariant, which describes the package version as the one "every entry point checks so old bytecode stops after an upgrade" (`blast_fun.move:4-5`). The proven specification of `share` confirms that its only abort condition is a launch that is already shared.

On Sui, a superseded package version stays callable, type identity survives a compatible upgrade, and one programmable transaction can call into different published versions. Once a second generation exists, a creator can pair the current, gated `launch::new` with a superseded `finalize` on the same initializer. `finalize` reaches `launch_ticket::destroy` (`launch_ticket.move:115-122`), which re-checks the ticket's supply and sell-burn rate against the constants compiled into that older bytecode. A stale `finalize` therefore validates against a stale ceiling, and the resulting `sell_burn_bps` is stored on the `Launch` with no setter.

Several factors bound the harm today:

- the package has never been published, so no superseded generation exists;
- [ADR 0049](#) places the ticket-policy re-check in the gated `launch::new`, so a tightening upgrade that follows the ADR closes the concrete harm without touching `finalize`;
- the affected `sell_burn_bps` is emitted on chain in `LaunchTicketCreated` before any trade, so counterparties can see it.

Impact. A defense-in-depth gap: a guard that the module's own convention calls for is missing, which after a future upgrade would let a creator finalize under superseded validation rules. It is not a demonstrated exploit.

Recommendation. Before the first publish, either thread `&PackageVersion` into `finalize` and `share` and call `assert_current()` first, matching every other launch-mutating function, or move the ticket supply, sell-burn, opening-round and symbol re-checks into the gated `launch::new`, as ADR 0049's intake-hardening section specifies. Gate the `LaunchInitializer` setters as well, for consistency with the stated invariant.

Remediation note. Every function named in the finding is now gated. `treasury_cap_mut` and `launch_ticket::new` stay ungated. That is acceptable because both are witness-gated: `launch_ticket::new` requires the Meme's one-time witness, and `treasury_cap_mut` requires the starting adapter's witness.

fees::new accepts Meme == Quote, leaving FeeLedger's four-bucket separation to the caller

I-1

INFORMATIONAL

Remediated

Location: `fees::FeeLedger`

Description. `fees::new` ([fees.move:24-32](#)) is the only constructor of `FeeLedger`, and it does not check that `Meme` and `Quote` differ. Both type parameters are `phantom` ([fees.move:13](#)) and appear only inside `Balance` types, so the type system does not require them to be distinct. `FeeLedger<Meme, Meme>` is a well-formed type, and constructing one would collapse the four trade-fee balances into two indistinguishable pairs of `Balance<Meme>`.

The distinctness that the layout assumes is enforced one level up:

`launch::new` ([launch.move:90-93](#)) rejects the pair with `invalid_meme_quote_pair`, a deliberate intake check recorded in ADR 0049. That guard covers every production path today:

- `fees::new` is `public(package)`, and its only caller is `launch::finalize` ([launch.move:262](#));
- `finalize` consumes a `LaunchInitializer`, a hot potato with no abilities whose only constructor is inside the asserting `launch::new` ([launch.move:101-114](#)).

Even a collapsed instance would lose nothing. The fields stay distinct, `accrue` joins each half only to its own balance, and the only visible effect is that each claim returns two coins of the same type to the same owner.

Impact. No reachable effect today. The type's own boundary does not enforce what its layout assumes, so a future in-package caller of `fees::new` would silently receive a collapsed instance.

Recommendation. Add the check as the first statement of `fees::new`, reusing the existing error macro:

```
assert!(
  type_name::with_original_ids<Meme>() != type_name::with_orig
  blast_fun::errors::invalid_meme_quote_pair!(),
);
```

Keep the change body-only. The launchpad's boundary script pins the exact signature of `new` and the struct's field list ([check_source_boundaries.sh:659-662](#)), and an `assert!` changes neither. Keep the guard in `launch::new` too: it rejects the pair earlier, at intake, with the same error code.

errors_tests registry pins only 35 of the 38 error macros

I-2

INFORMATIONAL

Remediated

Location: `blast_fun::errors`

Description. `errors.move` is the launchpad's sole error registry: abort codes 0 to 37, each declared as a `#[test_only]` constant and a matching `macro public(package)` function. The registry test [errors_tests.move](#) pins

each code's numeric value with `assert_eq!`, but it misses the codes added with the creation fee and per-launch partners ([errors.move:232-236](#)):

CODE	MACRO	RAISED IN
35	<code>invalid_creation_fee</code>	<code>launch::finalize</code> (launch.move:211)
36	<code>partner_not_accepted</code>	<code>launch_config::assert_partner</code> (launch_config.move:229 , launch_config.move:231)
37	<code>partner_mismatch</code>	<code>launch::claim_protocol_fees</code> and <code>launch::claim_protocol_fees_with_partn</code> (launch.move:621 , launch.move:641)

Nothing else holds these values in place:

- the `expected_failure` tests that exercise them name the constant while the abort comes from the macro, so they catch the two drifting apart, but not a renumbering that changes both;
- the boundary script has no registry-completeness rule, and no other test, including the Rust PTB harness, pins these codes.

A mutation run confirms the gap. Rotating the three codes in both constants and macros leaves every launchpad Move test passing, while the same kind of swap on two pinned codes fails the registry test. Only a developer edit to `errors.move` can trigger this; there is no user or attacker path.

Impact. No on-chain effect: today the code aborts with the documented values. The risk is a future renumbering, which the repository's contributor conventions make routine, since deleting an error renumbers every later code. Such a change to these codes would pass CI and silently change the abort numbers that off-chain tooling decodes. The PTB harness already matches aborts by numeric code ([lib.rs:499](#)), so these failures would be reported as the wrong errors.

Recommendation. Pin `invalid_creation_fee`, `partner_not_accepted` and `partner_mismatch` at 35, 36 and 37 in `errors_tests.move`, so that every code in `errors.move` again has an exact value check. To keep the next new code from slipping through, enforce completeness rather than relying on convention, as the count-checked registry test in

`sui/boosts/core` and the `check_error_registry()` rule in the `sui/boosts/aftermath_adapter` boundary gate already do.

Stale QuoteSet doc describes a removed virtual-reserve field

I-3

INFORMATIONAL

Remediated

Location: `blast_fun::events`

Description. The doc comment on `QuoteSet` (`events.move:26`) reads "A Quote's virtual reserve for future launches; zero means disallowed." The struct carries no reserve:

- it is `QuoteSet { quote: TypeName, allowed: bool }` (`events.move:27-30`);
- its only emitter, `events::quote_set`, publishes exactly the Quote's type name and the flag (`events.move:234-236`);
- its only caller, `launch_config::set_quote_allowed` (`launch_config.move:76-88`), adds or removes the Quote's key in `LaunchConfig.quotes` and forwards the flag.

The sentence is left over from the design that [ADR 0092](#) replaced, in which an administrator set one virtual reserve per Quote type and zero disallowed it. The reserve now belongs to each launch: the creator chooses it through `launch::set_virtual_quote` (`launch.move:131-136`), and `finalize` requires it to be non-zero (`launch.move:216`). Both clauses of the comment are therefore wrong: there is no reserve value, and disallowing a Quote removes its key rather than storing zero.

Impact. Documentation only. The emitted values are correct and the event's BCS layout is pinned by tests. The cost is a misleading description for SDK, indexer and dashboard authors who read the event vocabulary; the presale package documents the identical struct correctly (`creation_policy.move:127`).

Recommendation. Rewrite the comment to describe what the event records, for example: "A Quote was admitted to (`allowed`) or removed from future launches." No code, layout or migration change is needed, and the event's BCS shape must not change.

ProtocolAllocationSet doc implies the new share applies to existing launches

I-4

INFORMATIONAL

Remediated

Location: `blast_fun::events`

Description. The doc comment on the `ProtocolAllocationSet` event (`events.move:9`) describes the emitted value as "Blast's one-time fee share of every launch's Meme supply and final Quote reserve". The event is emitted by `launch_config::set_protocol_allocation` (`launch_config.move:38-59`), which updates only the `LaunchConfig` policy:

- each launch copies that policy into its own `allocation_plan` once, in `finalize` (`launch.move:215`);
- nothing rewrites a launch's plan afterwards, and `migrate` apportions from the stored plan (`launch.move:681`).

A new allocation therefore applies only to launches finalized after the event, as the setter's own doc says. The sibling events `LaunchTradeFeesChanged` and `CreationFeeChanged` state that future-launch scope.

Impact. No on-chain impact. An indexer or integrator that trusts this doc could attribute the new protocol share to launches that are already live, and so misreport their graduation payouts.

Recommendation. Reword the comment to future-launch scope, matching the setter, for example: "Blast's one-time fee share of the Meme supply and final Quote reserve of launches finalized from now on; finalized launches keep the share in their `LaunchCreated.allocation_plan`." No code or event-layout change is needed.

metadata_cap_id names two different cap types across launchpad events

I-5

INFORMATIONAL

Remediated

Location: `events::CreatorCapsClaimed`

Description. In `blast_fun::events`, the field name `metadata_cap_id` refers to two different object types:

- in `CreatorCapsClaimed` (`events.move:57-61`) and `MetadataCapReassigned` (`events.move:69-73`) it is the

`BlastMetadataCap` minted for the creator ([creator_authority.move:74-110](#)), the live metadata authority under [ADR 0082](#);

- in `LaunchTicketCreated` ([events.move:84-92](#)) it is the native Coin Registry `MetadataCap` ([launch_ticket.move:64-71](#)), which stays inside the launch and is not the authority.

Both fields carry the correct values, and a BCS test pins the exact bytes ([launch_tests.move:944-948](#)), so the defect is the name, not the data. The `Launch` struct already tells the two apart, as `metadata_cap` and `blast_metadata_cap_id` ([launch.move:52-55](#)); the event payloads are the only place where the names collide.

Impact. [ADR 0082](#) requires indexers to track the live metadata authority from `CreatorCapsClaimed` and `MetadataCapReassigned`, so this field is load-bearing for exactly that purpose. An SDK or indexer schema that keys or joins on the field name can conflate the immovable native cap with the live Blast cap. No value moves and no on-chain behavior changes. The name is a published ABI commitment ([ADR 0059](#)), but the lineage is still pre-publication ([ADR 0088](#)), so fixing it now is cheap; later it becomes an event-schema migration.

Recommendation. Rename the field to `blast_metadata_cap_id` in `CreatorCapsClaimed` ([events.move:60](#)) and, for consistency, in `MetadataCapReassigned` ([events.move:71](#)), matching the `Launch` field and the cap type it carries. The change touches the two emitters ([events.move:265-271](#), [events.move:277-279](#)), their call sites ([launch.move:334](#), [launch.move:377](#)) and the BCS test, and it must land before the [ADR 0059](#) freeze. Renaming the `LaunchTicketCreated` field instead is equally valid, since what must go is the collision. If the name is kept, the SDK and indexer schema must document that `metadata_cap_id` is type-scoped.

Formal Verification

Specifications: 66

Invariants and helper predicates: 17

MODULE	COUNT	SPECIFICATIONS
allocation_specs	8	apportion_fee_split_spec, apportion_spec, plan_spec, recipient_struct_role_consistency
bps_specs	2	apply_up_spec, value_spec
creator_authority_specs	6	claim_creator_caps_spec, curve_fee_cap_id_field_nested_life, destroy_curve_fee_cap_spec, destroy_new_account_spec, new_address_spec
launch_config_specs	11	add_partner_function_state_correct, remove_partner_function_state_correct, remove_partner_spec, set_creation, set_meme_decimals_allowed_spec, set_protocol_allocation_spec, set_quote_allowed_function_state_correct, set_quote_allowed_spec, set_trade
launch_specs	32	assert_curve_fee_cap_spec, buy_spec, claim_creator_fees_spec, claim_protocol_fees_with_partner_spec, deliver_opening_round_function_arguments, deliver_opening_round_function_state, deliver_opening_round_spec, final_launch_id_field_nested_lifecycle, launch_opening_round_quote_field, lp_fee_split_spec, migrate_spec, reassign_curve_fee_cap_spec, reassign_spec, set_allocation_plan_spec, set_description, set_graduation_target_spec, set_id, set_name_spec, set_opening_round, set_partner_function_state_correct, set_virtual_quote_spec, settle_operation, treasury_cap_mut_spec
launch_ticket_specs	1	new_spec
package_admin_specs	4	authorize_upgrade_spec, commit_upgrade, make_immutable_spec, wrap_spec
runtime_config_specs	2	set_creator_claim_signer_spec, set

We specify the launchpad's entry points in Move and prove the specifications against the code. The proofs follow the guarantees each party relies on: traders need value never to leak from a trade, creators and Blast need each claim to pay exactly its owner, bidders need an opening round to strand nothing, and every party needs configuration to stay bounded and upgrade authority to stay sealed. The table lists every specification; the properties whose proofs are not yet complete are named in [Coverage and Limitations](#), and this report claims nothing about them.

Exact, conserving sells. The constant-product curve over the virtual plus real Quote reserve is the protocol's pricing mechanism, and the sell path is specified to the atom and proven. Let m be the Meme sold, f_m and f_q the Meme and Quote fee rates, b the launch's sell-burn rate (all rates in basis points), R_m the Meme reserve and $V + R_q$ the virtual plus real Quote reserve. A sell then computes

$$\phi_m = \lceil \frac{mf_m}{10^4} \rceil, \quad \beta = \lceil \frac{(m - \phi_m)b}{10^4} \rceil, \quad q = \lfloor \frac{(V + R_q)(m - \phi_m - \beta)}{R_m + m - \phi_m - \beta} \rfloor$$

and pays the seller exactly $q - \lfloor qf_q/10^4 \rfloor$, while β joins the pending burn. A sell aborts exactly when the package generation is stale, the launch is not both shared and `Live`, the net output is zero or below the seller's minimum, or q exceeds the real reserve R_q . The virtual reserve therefore prices trades but is never paid out. The divisor $R_m + m - \phi_m - \beta$ is zero only when the reserve and the Meme reaching the curve are both zero, and such a sell pays nothing and aborts.

Nothing is created or lost in a sell. Writing F^P and F^C for Blast's and the creator's fee balances:

$$m = \Delta R_m + \beta + \Delta F_m^P + \Delta F_m^C, \quad -\Delta R_q = q_{\text{out}} + \Delta F_q^P + \Delta F_q^C$$

Each fee ϕ splits with Blast's live share π rounded up, so $\Delta F^P = \lceil \phi\pi/10^4 \rceil$, and the creator takes the exact remainder. A sell also preserves the launch invariant: the real reserve stays within the graduation target, the fees and the sell burn stay within their caps, and the opening-round balances stay consistent.

Conserving buys, and live launches unaffected by delisting. A scenario proves that removing a Quote from the admitted list leaves a live launch that trades in it untouched: its fees, virtual reserve and graduation target keep their snapshotted values. A buy that completes afterwards conserves both coins exactly, honors the buyer's minimum, strictly grows the real reserve and strictly shrinks the Meme reserve:

$$Q_{\text{in}} = Q_{\text{refund}} + \Delta R_q + \Delta F_q^P + \Delta F_q^C, \quad -\Delta R_m = M_{\text{out}} + \Delta F_m^P + \Delta F_m^C$$

Claims pay exactly their owner. Each launch holds its fees in its own `FeeLedger`, released only through capability- or role-gated claims, and each claim is proven to drain exactly its owner's balances and nothing else:

- the creator's claim requires the curve-fee cap the launch currently records, and pays exactly the creator's Meme and Quote balances while leaving the reserves, Blast's balances and the creation fee untouched;
- Blast's claim, gated by `TreasuryAdmin`, pays exactly Blast's Meme, Quote and SUI balances, and aborts on a launch that names a partner;
- the partner claim aborts unless the partner passed is the one the launch named, and splits each balance x with no rounding loss, where s is the partner's share:

$$x = x_{\text{Blast}} + \lfloor \frac{x s}{10^4} \rfloor$$

Every claim aborts when there is nothing to pay. The rounding primitive under every fee, `bps::apply_up`, is proven to compute $\lfloor ar/10^4 \rfloor$ and never to exceed a , for any rate r up to 10,000 bps.

Pro-rata opening rounds that strand nothing. The opening round pools bids into one pot that settles with a single curve buy. Joining adds exactly the bid to the bidder's entry, to the total of bids and to the pot, so the pot and the bids move together. Delivery sends bidder i , whose bid b_i is part of the undelivered bids B , exactly

$$\text{meme}_i = \lfloor \frac{M b_i}{B} \rfloor, \quad \text{quote}_i = \lfloor \frac{Q b_i}{B} \rfloor$$

of the Meme M and refundable Quote Q that settlement left, and touches no reserve or fee balance. The divisor B includes the bidder's own bid, which joining forces to be positive, and delivery aborts if B is zero.

Scenarios show that rounding dust never strands:

- with two bidders, the second delivery takes the exact remainder and leaves the bids, the Meme and the Quote all at zero;
- a single bidder's join, settlement and delivery end with every opening-round balance at zero;
- after migration, the last bidder's delivery still leaves every opening-round balance at zero.

Exact apportionment at graduation. Allocation plans and LP-fee splits divide amounts by basis points using largest-remainder apportionment. For a plan whose Meme shares sum to 10,000 bps, the curve's, pool's, Blast's and recipients' amounts sum exactly to the supply S . An LP-fee split divides each collection T exactly, each part getting at least its floor share and at most T :

$$C + L + P + \sum_i r_i = S, \quad c + a + w + u + \sum_j d_j = T, \quad \text{each part} \geq \lfloor \frac{T}{10000} \rfloor$$

The divisor is the constant 10,000. The constructors are proven too. `plan` aborts exactly when the curve's or the pool's Meme share is zero, and it leaves Blast's shares at zero for `finalize` to add. `recipient` and `route` reject only the zero address, and a plan that lists one destination twice keeps both entries.

Migration empties the launch. A scenario proves that every successful migration of a launch whose plan is balanced, which `finalize` requires, leaves the curve reserve, the real Quote reserve, the pending burn and the allocated Meme all at zero, with the launch `Migrated`.

One-time creator claim, revocable caps. Creator authority is a one-time claim that mints two capabilities, which the launch then recognizes by ID. Claiming aborts exactly when the generation is stale, the launch is not yet shared, the claim was already used, or the claimant fails the creator check. On success the launch records exactly the two new cap IDs, nothing else changes, and the launch invariant holds.

- Reassignment by `CreatorAdmin` sends a fresh cap to a non-zero recipient and retires any pending claim without touching a balance.
- A scenario shows that destroying the live curve-fee cap leaves every fee in place, and that a later reassignment records a fresh cap for the recipient without moving any fee.
- Metadata updates require the live `BlastMetadataCap`, and each one changes only its own field of the coin's registry entry.

Bounded, role-gated configuration. The configuration setters enforce hard bounds on the rates they set, where α_m and α_q are Blast's one-time allocation of the Meme supply and of the final Quote reserve:

$$f_q \leq 1,000, \quad f_m \leq 1,000, \quad \alpha_m \leq 9,998, \quad \alpha_q \leq 9,999 \quad (\text{bps})$$

Each administrative setter accepts the witness of exactly one role. The proofs pin down when each setter aborts, and that it changes only its own setting. Every administrative setter also aborts on a stale package generation.

ENTRY POINT	ROLE	ABORTS EXACTLY
<code>set_trade_fees</code>	<code>FeeAdmin</code>	either fee exceeds 9,998 bps, or neither fee changes
<code>set_creation_fee</code>	<code>FeeAdmin</code>	the fee is unchanged
<code>set_protocol_fee_share</code>	<code>FeeAdmin</code>	the share is unchanged
<code>add_partner</code>	<code>FeeAdmin</code>	the partner is already accepted indefinitely
<code>remove_partner</code>	<code>FeeAdmin</code>	the partner is not currently accepted indefinitely, or the expiration of its notice period would overflow a <code>u64</code>
<code>set_protocol_allocation</code>	<code>TreasuryAdmin</code>	the Meme share exceeds 9,998 bps, the Quote share exceeds 9,998 bps, or nothing changes
<code>set_quote_allowed</code>	<code>ListingAdmin</code>	the Quote is already in the requested state
<code>set_meme_decimals_allowed</code>	<code>ListingAdmin</code>	the decimals are already in the requested state
<code>set_migration_witness_allowed</code>	<code>AdapterAdmin</code>	the witness type is defined at <code>0x1</code> or <code>0x2</code> or is already in the requested state
<code>set_creator_claim_signer</code>	<code>ClaimSignerAdmin</code>	the key is neither empty nor 32 bytes, or is unchanged

A scenario proves the partner acceptance window. Adding a partner accepts it indefinitely; removing it keeps it accepted only for the notice

period that `blast_partners` sets; adding it again cancels a pending removal. Another scenario proves that the last partner a creator names before `finalize` is the one the launch records.

Sealed upgrade authority and generation cutover. Once `wrap` runs, upgrade authority sits in a shared `PackageAdmin` that seals the package's `Publisher` and `UpgradeCap`, and every transition is proven:

ENTRY POINT	ABORTS UNLESS	PROVEN EFFECT
<code>wrap</code>	the <code>Publisher</code> is from <code>blast_fun</code> , and the <code>UpgradeCap</code> is the original package's and still at version 1	—
<code>authorize_upgrade</code>	the generation is current, the cap is sealed, the digest is 32 bytes, and no upgrade is in flight	a ticket for this package, under the cap's current policy and for exactly that digest
<code>commit_upgrade_and_advance</code>	an upgrade is in flight, the receipt matches the sealed cap, the cap's native version is below the <code>u64</code> maximum, and the generation is current	the generation advances by exactly one, the cap stays sealed, and custody records the new package
<code>make_immutable</code>	the cap is sealed and the generation is current	custody no longer holds an upgrade cap; the generation and the <code>Publisher</code> are unchanged

None of these conditions involves time, which is why custody of the `UpgradeAdmin` role carries the whole safeguard (L-3). `wrap` rejects any cap that was upgraded first (L-1). The specifications of `share` and of the initializer setters confirm that those functions have no generation check (L-5).

Trusted Dependencies

DEPENDENCY	KIND	ADDRESS
blast_admin	package	—
blast_fun_timelock	package	—
blast_partners	package	—
Sui Framework	framework	0x0000000000000000000000000000000000000000000000000000000000000000
Move Standard Library	framework	0x0000000000000000000000000000000000000000000000000000000000000000

The launchpad's guarantees, including every proven property, hold only relative to the dependencies below, which our review takes as correct.

DEPENDENCY	TRUSTED FOR
blast_admin	The shared ACL, which issues an AdminWitness for a role only to a live AdminCap holding that role and delays root transfers; the operational generation behind PackageVersion; sealed custody of the launchpad's Publisher and UpgradeCap, and the native upgrade transitions
blast_partners	Partner terms read at claim time: pay sends a partner at most its share of Blast's fee, rounded down, and leaves the rest; shares are capped at 80%; delay_ms sets how long a removed partner stays accepted
blast_fun_timelock	The delays that blast_admin and blast_partners place on root transfers and partner changes
Sui Framework	Conservation of Balance and Coin values, Coin Registry supply modes and metadata, dynamic fields, package upgrade tickets and receipts, Ed25519 verification, the clock and object transfers

DEPENDENCY	TRUSTED FOR
Move Standard Library	Type identities through <code>type_name</code> , used for the Meme and Quote distinctness check and for witness checks; checked <code>u64</code> arithmetic, including <code>mul_div</code> and <code>mul_div_ceil</code> ; options, vectors and BCS encoding

Two dependencies carry the most weight. Every role-gated result assumes that the ACL hands a role's witness only to an administrator that holds the role, so the access-control guarantees are exactly as strong as the ACL and the custody of its root (L-4). The partner-claim proof assumes that a partner's share never exceeds 100% of Blast's fee, which `blast_partners` enforces with its tighter 80% cap.

Coverage and Limitations

This report establishes properties of `blast_fun` at the review start commit, relative to the dependencies in [Trusted Dependencies](#). It does not establish the following.

Specified but not yet proven. These properties have written specifications whose proofs are not complete, so the report makes no claim about them:

AREA	NOT ESTABLISHED	PROVEN NEAR
Curve buys	The pricing, refund, abort conditions and graduation transition of <code>launch::buy</code> , including the cap that stops a buy at the graduation target; L-2 sits on this path	A completed b conserves Quc and Meme exactly
Opening-round settlement	The single pot buy made by <code>launch::settle_opening_round</code>	Joining, pro-ra delivery, and the full join, settle and deliver lifecycle
Launch finalization	That <code>launch::finalize</code> enforces every setting check, takes exactly the creation fee, splits the supply between the curve and the allocations, and starts a well-formed launch	Each initializer setter; the last partner named the one record

AREA	NOT ESTABLISHED	PROVEN NEAR
Migration payouts	The amounts <code>launch::migrate</code> pays each recipient, accrues for Blast and returns to the adapter	Migration leave every launch balance at zero
Quote apportionment	That <code>allocation::apportion_quote</code> and the shared routine <code>allocation::apportion</code> split their totals exactly	Exact splits of Meme supply and of LP-fee collections
Account-claim authentication	The exact abort conditions of <code>creator_authority::claim_creator_caps</code> : the attestation parse, the expiry check and the signature check	—
Coin creation	That <code>launch_ticket::new</code> caps the sell burn at 50% and mints exactly the treasury's supply	—

Assumptions of the proofs. The sell and creator-claim proofs take a well-formed launch as a precondition and prove that it stays well-formed; that a new launch starts out well-formed is part of the finalization property above. The sell proof also assumes that a launch's balances of one coin fit in a `u64`, as any coin's total supply does.

Upgrades. The proofs describe the reviewed bytecode. A governed upgrade, which needs no delay, can replace any of it (L-3, L-4). After an upgrade, the functions without a generation check stay callable in the superseded version (L-5).

Outside the package. This review does not establish what an admitted adapter does with the pool's share and the LP-fee split after `migrate`, nor how the root, upgrade and claim-signer keys are held.

Methodology

Review. We review the `blast_fun` sources manually and with tooling, line by line, against the repository's decision records and tests, examining every function, type and state transition in scope. We check each candidate issue against the code and report only confirmed issues, with their severity set by impact and by who can trigger them.

Specification. Properties are written as Move specifications in a separate `specs` package, using the Sui Prover's specification language. Each specification states a function's preconditions, its full effect (including what it leaves unchanged), and the exact conditions under which it aborts. Scenario specifications chain public calls, such as join, settle and deliver, or delist and then buy, to prove end-to-end properties that no single function states.

Proof. A specification counts as proven only when its machine-checked proof passes in the Lean theorem prover against the current code. Specifications that are written but not yet proven are reported as such in [Coverage and Limitations](#), never as proven.

Ghost state and composition. Ghost state records where objects are transferred, so the proofs can state who receives a reassigned capability or an opening-round delivery. Specifications are factored the way the code is: shared building blocks are specified once and reused when their callers are proven, and loops carry explicit loop invariants.

Commits

REVISION	COMMIT
Upstream protocol revision	b549bb16e02e4c6e8ff59e1958435a638c784973
Verified revision	d3e53727c74d178193a4d5baaa2f398c1fa47bee
Remediation merge	803ebfc67f037526081715df1deaf5c0dfe6fcf2
Final reviewed commit (remediation review)	01d7659617abcd552cece6d05b9db409ec4ff2c0
Fix for L-1	e77c36d6 , d184dd22 and 9e18c028
Fix for L-2	539b62a7 and 3fa3ce2a
Fix for L-5	22fe5363
Fix for I-1	d42e0da3
Fix for I-2	a8f239d2

REVISION	COMMIT
Fix for I-3	39228b70 and bf7ed780
Fix for I-4	f716053a
Fix for I-5	416bfa23

This report reflects our review of the code changes at the commits and on-chain versions noted above. A review reduces the risk of certain classes of bugs but does not guarantee the absence of all vulnerabilities; only the reviewed changes and properties are covered. This report is provided for informational purposes only, not as financial, legal, or investment advice. No liability is assumed for losses arising from reliance on this report. Users should perform their own due diligence before interacting with any smart contract system.