

IMMUNEFI AUDIT COMPETITION

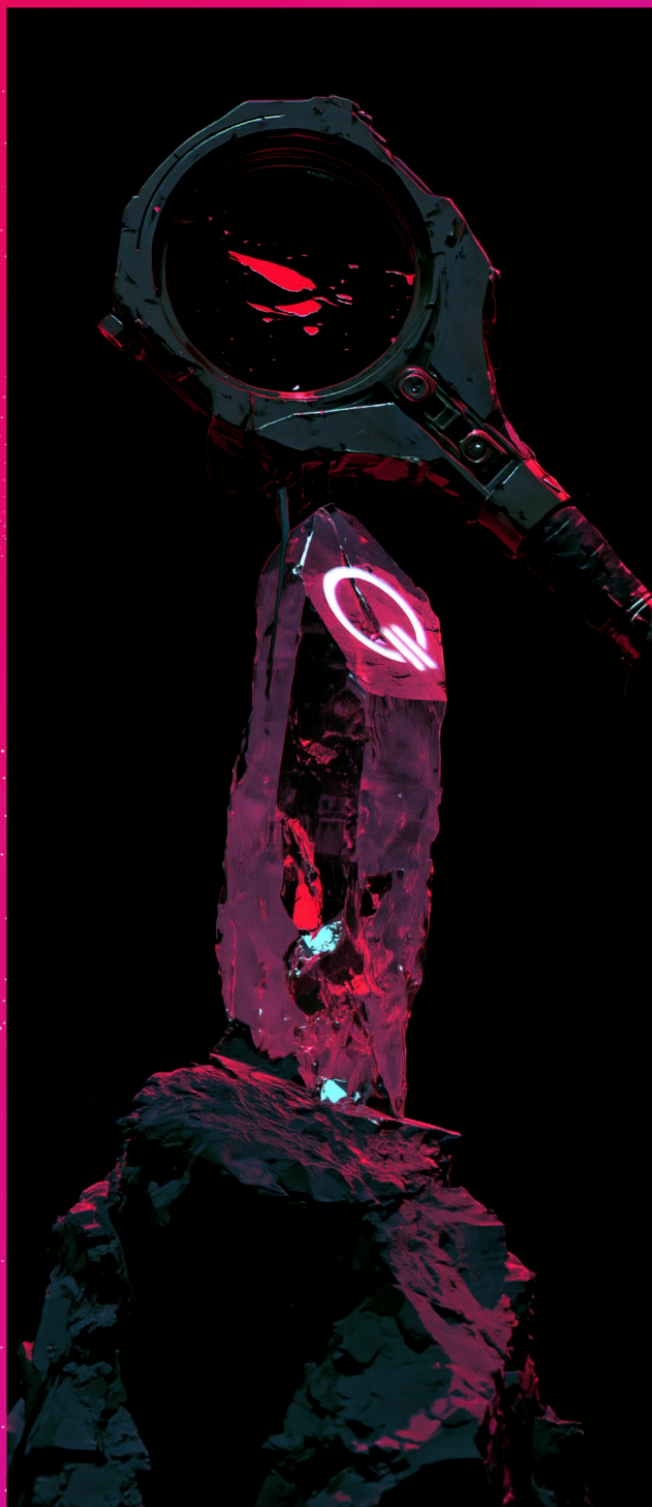
 Immunefi /  QUANTUS

DATE Aug. 12, 2026

TOP AUDITORS
bytes
zealynx
Valisthea

REPORT BY Immunefi

01	Overview
02	Scope of assets
03	Summary
04	Findings



Contents

About Immunefi	3
Terminology	4
Executive Summary	6
Findings — Blockchain/DLT	13
Findings — Websites & Apps	77

About Immunefi

Immunefi is the security platform for the onchain economy, and 93% of crypto's critical vulnerability disclosures now flow through it. Founded in December 2020, it protects more than \$180 billion in assets across 650-plus protocols, including MakerDAO, Optimism, Filecoin, LayerZero, Chainlink, Arbitrum, Lido, and Polygon. It has helped prevent more than \$25 billion in hacks and paid more than \$140 million in bounties, more than the rest of the market combined. Its unified security platform combines bug bounties, audit competitions, AI-assisted security, and coordinated vulnerability disclosure, powered by more than 85,000 researchers and a proprietary exploit dataset built since 2020. The thesis: crowdsourced security, supercharged with AI, is the only defense that scales to meet AI-driven attacks.

Learn how you can benefit too at immunefi.com.

Immunefi Audit Competitions

Immunefi has one of the largest communities of security researchers, with over 85,000 members. This community is mobilised through Audit Competitions.

In audit competitions, a prize pool is shared, and the codebase is opened to Immunefi's researcher network for a fixed competition window. Audit Competitions are quantitatively superior to private audits, and these findings are supported by data from 1,178 public audit reports from Trail of Bits, Zellic, and OpenZeppelin.

1. Improving the cost efficiency of bug finding: On average, a critical or high vulnerability costs about \$66,000 in a private audit. This cost is reduced by about 10x in an audit competition, with an average of \$6,548.

2. Improving coverage, scrutiny and findings: An audit assigns 2 to 5 people to your code, while recent Immunefi competitions drew 952, 943 and 1,415 submissions. Hundreds of researchers cover more of the codebase and find more bugs.

The median Immunefi audit competition finds 2 serious findings, compared to 0 for a Tier-1 private audit. The same applies for critical bugs: more than 1 in 2 Immunefi audit competitions find those, compared to only 1 in 6 Tier-1 audits. All in all, Immunefi audit competitions average 6.2 critical or high findings, and Tier-1 audits only about 1.5.

Comparative Analysis between Private Audits and Audit Competitions

	Tier-1 private audit	Immunefi audit competition
Engagements measured	1,178	58
Median serious findings	0	2
Average per engagement	1.5	6.2
Surfaced at least one critical	1 in 6	more than 1 in 2
Cost per critical-or-high	~\$66,000	\$6,548

The best single report across all 1,178 found 21 critical-and-high issues. One Immunefi competition on a \$100K pool found 43. Recent competitions on Immunefi include Firedancer V1, Base Azul, and the Ethereum Protocol Attackathon. Every finding, severity and payout is published at immunefi.com/audit-competition.

Terminology

For the purpose of this assessment we adopt the Immunefi severity classification defined by the Quantus Audit Competition programme. Severity is determined by impact: each finding is mapped to a named impact from the programme's Impacts in Scope table, and the severity of that impact is the severity of the finding.

The table is defined per asset type, so the same consequence can carry a different severity depending on whether it is reached through the chain, an application, or a smart contract. A runnable proof of concept was required for every submission at every severity level. Insight is a tier below Low, used for observations with no demonstrated in-scope impact; it cannot be self-selected at submission and is designated by Immunefi or the project team.

The bands below are those that apply to the assets in scope of this competition.

Insights aren't included in the table, since they lack severity/impact.

Blockchain / DLT

Severity	In-scope impacts
Critical	· Network not being able to confirm new transactions (total network shutdown) · Unintended permanent chain split requiring hard fork · Soundness bug in the ZK circuit · Direct loss of funds · Inflation bugs (unauthorized minting) · Permanent freezing of funds
High	· Unintended chain split (network partition) · Temporary freezing of network transactions by delaying one block by 500% or more of the average block time of the preceding 24 hours · Causing network processing nodes to process transactions from the mempool beyond set parameters · RPC API crash affecting projects with 25% or more of the market capitalization
Medium	· Increasing network processing node resource consumption by at least 30% without brute force actions · Shutdown of 30% or more of network processing nodes without brute force actions, but not shutting down the network · A bug in the respective layer 0/1/2 network code resulting in unintended smart contract behavior with no concrete funds at direct risk
Low	· Shutdown of more than 10% but less than 30% of network processing nodes without brute force actions · Modification of transaction fees outside of design parameters

Websites & Applications

Severity	In-scope impacts
Critical	· Execute arbitrary system commands · Retrieve sensitive data or files from a running server · Taking down the application or website · Taking state-modifying authenticated actions on behalf of other users without any interaction by that user · Subdomain takeover with already-connected wallet interaction · Direct theft of user funds · Malicious interactions with an already-connected wallet · Injection of malicious HTML or XSS through metadata
High	· Injecting or modifying persistent static content without JavaScript · Changing sensitive details of other users without already-connected wallet interaction and with up to one click of user interaction · Improperly disclosing confidential user information · Subdomain takeover without already-connected wallet interaction
Medium	· Changing non-sensitive details of other users with up to one click of user interaction · Injecting or modifying reflected static content without JavaScript · Redirecting users to malicious websites (open redirect) · Inaccuracies to the weight accounting system that under- or overcharge by more than 50% of the correct amount
Low	· Inaccuracies to the weight accounting system that under- or overcharge by less than 50% of the correct amount · Changing details of other users with significant user interaction · Taking over broken or expired outgoing links · Temporarily disabling a user's access to the target site

Executive Summary

Over the course of 13 days in August 12 – August 25, 2026, Quantus ran an audit competition on Immunefi covering seven assets across Blockchain/DLT and Websites & Applications. The competition resolved into **66 unique findings**: 9 Critical, 19 High, 26 Medium, 9 Low and 3 Insights.

Summary

Name	Quantus
Competition Type	Audit Competition
Competition Period	August 12 – August 25, 2026
Reward Pool	\$20,000 USDC
Languages	Rust, Dart (Flutter)
Scope	64,840 lines of code across seven assets
PoC Requirement	Runnable, required at every severity
Unique Findings	66

Assets in Scope

Asset	Repository	Commit
Quantus chain	immunefi-team/audit-comp-quantus-chain	3b243b87
Quantus mobile app	mobile-app">immunefi-team/audit-comp-quantus-apps → mobile-app	d5fc4d65
Quantus SDK	quantus_sdk">immunefi-team/audit-comp-quantus-apps → quantus_sdk	d5fc4d65
qp-zk-circuits	immunefi-team/audit-comp-quantus-qp-zk-circuits	2a6914ed
qp-rusty-crystals (hdwallet)	hdwallet">immunefi-team/audit-comp-quantus-qp-rusty-crystals → hdwallet	94dfe6e6
qp-rusty-crystals (dilithium)	dilithium">immunefi-team/audit-comp-quantus-qp-rusty-crystals → dilithium	94dfe6e6
qp-poseidon	Quantus-Network/qp-poseidon	f885ad5

Issues Found

Severity	Blockchain/DLT	Websites & Apps	Total
Critical	5	4	9
High	13	6	19
Medium	21	5	26
Low	7	2	9
Insight	2	1	3
Total	48	18	66

Findings index – Blockchain/DLT

Finding ID	Severity	Report Name
IMM-BC-CRITICAL-89136	Critical	Missing deposit-release calls in the high-security whitelist permanently freeze reserved balances, including a rescuer's recovery deposit
IMM-BC-CRITICAL-88628	Critical	<code>fungible::Mutate::transfer</code> checks the deposit precondition on a larger value than it credits, destroying near-dust senders' funds
IMM-BC-CRITICAL-89438	Critical	Miner-controlled future timestamps let a reversible transfer execute inside its cancellation window
IMM-BC-CRITICAL-88609	Critical	High-security gate inspects only the call, so an unbounded transaction tip drains a protected account in one block
IMM-BC-CRITICAL-89819	Critical	Missing FIPS 204 domain separation lets a victim's message signature authorize a transaction
IMM-BC-HIGH-89065	High	Wormhole deposits below the circuit's fee-exitable minimum are recorded but can never be exited, freezing user funds
IMM-BC-HIGH-88969	High	Multisig re-decodes opaque call bytes without a depth limit, letting free stack-exhaustion poison consume block authors
IMM-BC-HIGH-88250	High	Transient accept errors terminate the litep2p listener permanently, letting unauthenticated TCP floods remove a node from the network
IMM-BC-HIGH-88404	High	Scheduler drops the caller's preimage reference on a failed enactment retry, silently voiding a passed referendum
IMM-BC-HIGH-88938	High	Wormhole miner volume-fee credit records no ZK-tree leaf, stranding the keyless block author's share
IMM-BC-HIGH-88498	High	Mining rewards are minted at planck precision while the miner's only leaf commits whole quanta, stranding the remainder
IMM-BC-HIGH-88485	High	Wormhole pool deduplication keys on unverified nullifiers, letting one proof become many fee-free mempool entries
IMM-BC-HIGH-89062	High	Unsigned wormhole exit proofs are verified without limit per block, letting any peer exhaust every producer's build budget
IMM-BC-HIGH-89247	High	Wormhole inertness check keys on nullifiers instead of value, letting zero-value exits settle free and grow state

Finding ID	Severity	Report Name
IMM-BC-HIGH-88566	High	Bundle-level fee rounding settles less than the per-leaf fee the circuit locks, stranding the difference
IMM-BC-HIGH-88223	High	Wormhole miner fee and aggregator rebate are credited without a zk-tree leaf, freezing them permanently
IMM-BC-HIGH-89139	High	Credits to wormhole addresses are not quantized, so sub-quantum value is committed to no leaf and unexitable
IMM-BC-HIGH-88297	High	Scheduled transfers below the existential deposit can never execute and only the guardian can clear them
IMM-BC-MED-88219	Medium	Invalid proof-of-work is not attributed to the sending peer, so a hostile peer is never penalised
IMM-BC-MED-88224	Medium	Unvalidated peer-advertised block height pauses mining and suppresses block-import notifications
IMM-BC-MED-88226	Medium	<code>multistream-select</code> sizes its read buffer to an unauthenticated peer's declared frame length
IMM-BC-MED-88249	Medium	<code>ExtrinsicBaseWeight</code> still prices sr25519 after the switch to ML-DSA, undercharging every signed extrinsic
IMM-BC-MED-88218	Medium	Inherited sr25519 base weight leaves Dilithium verification unmetered in blocks and unpriced in the pool
IMM-BC-MED-88397	Medium	Self-directed reversible transfers record wormhole ZK-tree leaves for credits that never occur
IMM-BC-MED-90047	Medium	Unbounded SURI path depth in <code>derive_dev_account</code> bypasses the dev-account cap through the public light-client protocol
IMM-BC-MED-88227	Medium	Degenerate-key filter in ML-DSA <code>verify_var</code> misses the <code>tl</code> family that also drops the challenge term
IMM-BC-MED-88386	Medium	<code>txWatch</code> subscription tasks outlive unsubscribe, so RPC caps do not bound retained tasks
IMM-BC-MED-89154	Medium	Sparse public wormhole batches retain the full 742-exit weight declaration after settling one exit
IMM-BC-MED-90343	Medium	Txpool skips maintenance on every finality event, so watched transactions are falsely timed out
IMM-BC-MED-88736	Medium	Pre-authentication ML-KEM/ML-DSA Noise work is uncapped, letting unauthenticated peers inflate node CPU and memory
IMM-BC-MED-90210	Medium	Default transaction-pool size exceeds the gossip known-hash cache, causing repeated full-peer disconnects and resource amplification
IMM-BC-MED-88329	Medium	Missing guardian consent in <code>set_high_security</code> lets one signer exhaust any guardian's bounded quota
IMM-BC-MED-89524	Medium	<code>retarget_schedule</code> skips settlement inside the claim rate-limit window, reassigning the outgoing beneficiary's vested funds
IMM-BC-MED-90408	Medium	Safe RPC <code>state_call</code> exposes <code>GenesisBuilder</code> with an unbounded derivation template, pinning node CPU cores
IMM-BC-MED-88215	Medium	Any account can permanently consume another address's bounded guardian index with no consent or removal path

Finding ID	Severity	Report Name
IMM-BC-MED-88671	Medium	Permissionless vesting <code>claim</code> lets a third party fragment a grant into rounding-lossy payout leaves
IMM-BC-MED-89213	Medium	Missing account-reaping cleanup leaves dead entries permanently occupying a guardian's bounded index
IMM-BC-MED-89889	Medium	Permissionless vesting <code>claim</code> strands a schedule below the termination threshold, blocking the treasury's clawback
IMM-BC-MED-89307	Medium	Sub-existential-deposit block fees are minted before the account-endowing reward and diverted to the treasury
IMM-BC-LOW-88568	Low	Transaction payment weight omits the custom fee collector's storage access, undercharging every paid extrinsic
IMM-BC-LOW-89098	Low	<code>count_transfers</code> has no <code>Multisig::execute</code> arm, so wrapped transfers carry zero proof-recording weight
IMM-BC-LOW-90448	Low	<code>recover_funds</code> weight counts pending transfers but not the scheduler-agenda density its cancellations pay for
IMM-BC-LOW-90496	Low	Unreserved tips let one account fill a node's future queue and evict every lower-priority transaction
IMM-BC-LOW-88830	Low	Private-batch circuit does not bind a real positive-value output to a non-zero exit account
IMM-BC-LOW-89077	Low	Genesis wormhole leaves record <code>total_balance</code> , so reserved funds are represented as exitable value
IMM-BC-LOW-89310	Low	Asymmetric difficulty retarget lets an author-supplied timestamp book a deficit later blocks cannot repay
IMM-BC-INSIGHT-88243	Insight	<code>check_wormhole_path</code> validates only the coin type and the crate exports no path constructor
IMM-BC-INSIGHT-88290	Insight	<code>PeerId</code> is derived from the sender's raw protobuf bytes rather than from the decoded public key

Findings index – Websites & Apps

Finding ID	Severity	Report Name
IMM-WA-CRITICAL-89007	Critical	Lock screen leaves the underlying wallet in the accessibility tree, letting a bound service sweep the wormhole balance
IMM-WA-CRITICAL-88730	Critical	Generated <code>MultiAddress::Index</code> codec consumes a compact integer the runtime does not, hiding <code>transfer_all</code> in a multisig proposal
IMM-WA-CRITICAL-89999	Critical	Multisig discovery stores the indexer-supplied address without recomputing the derivation, letting a hostile indexer substitute the account
IMM-WA-CRITICAL-89802	Critical	Mobile signing path never validates the endpoint-supplied genesis hash, so a malicious RPC binds signatures to another chain
IMM-WA-HIGH-88348	High	<code>Utility.if_else</code> decoding describes its primary branch twice, letting a 213-byte proposal freeze the wallet UI isolate
IMM-WA-HIGH-88483	High	Approve sheet signs multisig bytes it failed to decode, showing amount zero and an empty destination
IMM-WA-HIGH-90460	High	Quantus POS latches <code>Payment Received</code> on a pre-execution pool notification the payer can still replace
IMM-WA-HIGH-89806	High	SDK <code>RuntimeCall</code> decoder lacks the chain's Vesting index, leaving an undecodable multisig approval enabled
IMM-WA-HIGH-89095	High	Android recovery-phrase clipboard expiry relies on an in-process timer that dies with the app
IMM-WA-HIGH-88914	High	Hide balances redacts amounts on the multisig approval screen, concealing an <code>if_else</code> fallback transfer
IMM-WA-MED-88274	Medium	<code>Utility.batch</code> decoding drops the nested transfer summary, hiding the amount on the multisig approval sheet
IMM-WA-MED-89188	Medium	UTXO pager orders only by block height, so a page boundary drops a same-height transfer from the claim list
IMM-WA-MED-88786	Medium	Point-of-sale screen promotes a pre-execution pool notification to paid, so a rolled-back <code>batch_all</code> reads as payment received
IMM-WA-MED-88221	Medium	Unvalidated <code>amount</code> in the <code>/pay</code> deep link freezes the wallet on unbounded <code>BigInt</code> materialisation
IMM-WA-MED-89478	Medium	Balance-privacy display blanks the transfer amount on the multisig approval sheet while Approve stays enabled
IMM-WA-LOW-88305	Low	<code>predictMultisigAddress</code> omits the pallet's signer deduplication and returns an address the chain never creates
IMM-WA-LOW-88263	Low	Late check-phrase lookup overwrites the current recipient's phrase, so the review screen vouches for another address
IMM-WA-INSIGHT-88382	Insight	Wallet mnemonic stored with the default iOS Keychain accessibility class

Findings – Blockchain/DLT

IMM-BC-CRITICAL-89136

Missing deposit-release calls in the high-security whitelist permanently freeze reserved balances, including a rescuer's recovery deposit

Report ID	#89136
Severity	Critical
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of funds

Description

Quantus ships two account-protection features that the runtime composes on purpose. High-security mode, entered through `ReversibleTransfers::set_high_security`, restricts an account to a closed whitelist of calls; the primitives crate describes that whitelist as including "only operations that are reversible or delayed" ([src/lib.rs#L82-L86](#)), and the pallet's own warning block tells users who no longer want the restrictions that they "can simply transfer their funds to a different account using `schedule_transfer`" ([src/lib.rs#L345-L367](#)). Social recovery is wired into the same policy deliberately: `pallet_recovery::Config` sets `type HighSecurity = HighSecurityConfig` ([configs/mod.rs#L511-L522](#)).

The policy predicate is `!is_high_security(who) || is_whitelisted(call)` ([src/lib.rs#L132-L147](#)), and `is_whitelisted` is a four-element `matches!`:

```
fn is_whitelisted(call: &RuntimeCall) -> bool {
    matches!(
        call,
        RuntimeCall::ReversibleTransfers(
            pallet_reversible_transfers::Call::schedule_transfer { .. }
        ) | RuntimeCall::ReversibleTransfers(pallet_reversible_transfers::Call::cancel { .. }) |
        RuntimeCall::ReversibleTransfers(
            pallet_reversible_transfers::Call::recover_funds { .. }
        ) | RuntimeCall::Vesting(pallet_vesting::Call::claim { .. })
    )
}
```

— [configs/mod.rs#L683-L693](#). It is enforced in the transaction extension and re-enforced at every origin-rewriting wrapper, including `Recovery::as_recovered` ([src/lib.rs#L450-L462](#)). Both recovery deposits are reserved balances: `create_recovery` reserves `ConfigDepositBase` plus `FriendDepositFactor` per friend, which this runtime sets to 10 UNIT plus 1 UNIT for each of up to nine friends ([src/lib.rs#L536-L539](#),

`configs/mod.rs#L500-L509`), and `initiate_recovery` reserves a 10 UNIT `RecoveryDeposit` on the rescuer (`src/lib.rs#L580-L582`). The calls that release them are `remove_recovery`, dispatchable only by the account itself (`src/lib.rs#L769-L781`), and `close_recovery`, which takes `ActiveRecoveries[origin][rescuer]` and is therefore dispatchable only as the rescued account (`src/lib.rs#L704-L712`).

Neither release call is whitelisted, and none of the four calls that are whitelisted decreases `reserved`. `schedule_transfer` holds from free balance (`src/lib.rs#L827-L831`); the closing sweep in `recover_funds` dispatches `transfer_all { keep_alive: false }`, which moves free balance only (`src/lib.rs#L608-L619`); `cancel` acts on a `ScheduledTransfer` hold; `Vesting::claim` is inbound. No enrolment order avoids the result, because `create_recovery` is not whitelisted either, so an enrolled account can never configure recovery, and `set_high_security` rejects a second enrolment (`src/lib.rs#L384-L387`), so the state cannot be unwound.

A user who configures recovery and then enrolls loses the configuration deposit permanently — 12 UNIT in the proof of concept's two-friend case, 10 to 19 UNIT at the runtime's parameters. The loss also falls on an uninvolved third party: a rescuer who completes the entire documented M-of-N flow against a high-security account cannot reclaim their own 10 UNIT, because the only call that releases it must be dispatched as the account the whitelist has gagged. The outstanding reserve pins a consumer reference, so the account cannot be reaped and the sweep is additionally forced to leave the existential deposit behind. This is permanent freezing of funds, roughly 29 UNIT plus the existential deposit per affected recovery, split across two parties, with no signed origin able to release it at any time.

Proof of Concept

1. The victim calls `Recovery::create_recovery` with two friends; 12 UNIT is reserved on the account.
2. The victim calls `ReversibleTransfers::set_high_security` naming a guardian; the call succeeds with the reserve outstanding and emits no warning about it.
3. `is_call_allowed` now returns false for both `Recovery::remove_recovery` and `Recovery::close_recovery` on that account.
4. A rescuer runs the documented flow — `initiate_recovery`, which reserves 10 UNIT on the rescuer, then two `vouch_recovery` calls, then `claim_recovery`.
5. Control: `as_recovered(lost, ReversibleTransfers::schedule_transfer { .. })` succeeds, so recovery works for moving free balance.
6. `as_recovered(lost, Recovery::close_recovery { .. })` and `as_recovered(lost, Recovery::remove_recovery { .. })` both fail with `CallNotAllowedForHighSecurity`; `close_recovery` dispatched by the rescuer directly fails with `NotStarted`, and `poke_deposit` refunds nothing.
7. The guardian's `recover_funds` leaves the victim's free balance at the existential deposit while 12 UNIT stays reserved, and a second `recover_funds` changes neither figure.

IMM-BC-CRITICAL-88628

`fungible::Mutate::transfer` checks the deposit precondition on a larger value than it credits, destroying near-dust senders' funds

Report ID	#88628
Severity	Critical
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Direct loss of funds

Description

`fungible::Mutate::transfer` in the vendored FRAME support tree is the transfer body every `Balances` extrinsic runs: `pallets/balances` own `impl fungible::Mutate` defines only the `done_*` event hooks and inherits the default body ([impl_fungible.rs#L191-L211](#)), and `transfer_allow_death` reaches it with `Expendable` ([src/lib.rs#L671-L680](#)). The intended behaviour for a sub-existential-deposit transfer to a dead destination is stated by the project's own test suite, which asserts `TokenError::BelowMinimum` for exactly that call shape ([dispatchable_tests.rs#L31-L45](#)), and the in-code comment states the accounting invariant the body is meant to hold.

The body evaluates its deposit precondition on one quantity and credits another:

```
let dust = Self::can_withdraw(source, amount).into_result(preservation != Expendable)?;
let actual = amount.saturating_add(dust);
Self::can_deposit(dest, actual, Extant).into_result()?;
if source == dest {
    // Nothing is debited and no reap occurs on the self path, so the reported amount must
    // not include the hypothetical dust.
    return Ok(amount)
}
```

```
let debited = Self::decrease_balance(source, amount, BestEffort, preservation, Polite)?;
...
```

— [regular.rs#L327-L342](#).

The comment's premise does not hold in this tree. `decrease_balance` returns `old_balance.saturating_sub(new_balance)` with `new_balance = old_balance - amount`, and disposes of dust separately through `write_balance` and `handle_dust`, so `debited` can never exceed `amount` ([regular.rs#L185-L197](#)). Meanwhile `increase_balance` with `BestEffort` returns `Ok(Default::default())` and writes nothing when the new balance falls below the minimum ([regular.rs#L205-L222](#)), and the caller

discards that result with `let _ =`. Whenever `amount < ED <= amount + dust`, the widened gate admits the call and the credit silently no-ops.

At the shipped parameters — `EXISTENTIAL_DEPOSIT = MILLI_UNIT = 1_000_000_000` planck ([src/lib.rs#L97-L102](#)) — a sender holding exactly one existential deposit who sends 500,000,000 planck is debited in full, the recipient stays at zero, and the call returns `Ok(500000000)`. `done_transfer` emits `Event::Transfer` for value that never moved ([impl fungible.rs#L204-L210](#)), so indexers that credit on events record a deposit that does not exist. Each incident also leaves `total_issuance` permanently above the sum of account balances by the transferred amount, and nothing on-chain reconciles it. The proof of concept measures a worst single loss of 1,375,000,000 planck, and destroys the entire balance of all four accounts that were in the vulnerable band on a fork of live Heisenberg state. `transfer_all` with `keep_alive = false` is a second entry point ([src/lib.rs#L738-L751](#)), and the multi-asset trait carries a verbatim clone of the same body ([regular.rs#L373-L396](#)). This is direct loss of funds for ordinary users sweeping near-empty accounts.

Proof of Concept

1. Fund a sender with exactly the existential deposit, 1,000,000,000 planck, and leave the destination account dead.
2. Call `Balances::transfer_allow_death` for 500,000,000 planck, half the existential deposit.
3. `can_withdraw` reports 500,000,000 planck of dust, so the deposit gate is evaluated on 1,000,000,000 planck and passes.
4. The call returns `Ok(500000000)`; the sender's balance goes to zero and the recipient's stays at zero.
5. `total_issuance` minus the sum of all account balances now equals the transferred amount, and the divergence persists.
6. Control: the same call from a sender funded at ten times the existential deposit is rejected with `Token(BelowMinimum)`, and `transfer_keep_alive`, which passes `Preservation::Preserve`, is rejected with `Token(NotExpendable)`.
7. Replaying the call against a fork of live Heisenberg state pinned at block #821811 destroys the full balance of each of the four accounts then sitting in the vulnerable band, 4,000,000,002 planck in total.

IMM-BC-CRITICAL-89438

Miner-controlled future timestamps let a reversible transfer execute inside its cancellation window

Report ID	#89438
Severity	Critical
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Direct loss of funds

Description

A reversible transfer holds the sender's funds and stays cancellable for the whole delay the sender chose; `cancel` is the revocation right the feature exists to provide, and the guarantee is meaningful only if a block producer cannot shorten that window. This runtime accepts a timestamp delay as short as one target block time, because `pallet_reversible_transfers::Config` sets `type MinDelayPeriodMoment = TargetBlockTime` and `TargetBlockTime` is `TARGET_BLOCK_TIME_MS = 12_000` ([configs/mod.rs#L534-L552](#), [configs/mod.rs#L150-L156](#)).

`schedule_transfer_with_delay` validates a timestamp delay against that minimum ([src/lib.rs#L681-L691](#)) and turns it into `DispatchTime::After(Timestamp(T::TimeProvider::now() + millis))`, reading the on-chain clock ([src/lib.rs#L776-L785](#)). Scheduler normalises that target into a `TimestampBucketSize` bucket of 24,000 ms and advances one bucket ([src/lib.rs#L678-L708](#)). Its `on_initialize` then services timestamp agendas using `T::TimeProvider::now()` ([src/lib.rs#L399-L420](#)), normalising the reading it finds and servicing every bucket up to it ([src/lib.rs#L1112-L1160](#)). Separately, `pallet_timestamp::check_inherent` accepts any claimed timestamp up to `MAX_TIMESTAMP_DRIFT_MILLIS`, a constant 30,000 ms, ahead of the importing node's own inherent data, with `MinimumPeriod` of 100 ms as the only lower bound ([src/lib.rs#L302-L327](#), [configs/mod.rs#L174-L184](#)). That check runs on the production import path — `import_block` calls `check_inherents` ([src/lib.rs#L247-L264](#)), whose height gate is never taken because the node passes `0` for `check_inherents_after` and supplies its real system clock through `sp_timestamp::InherentDataProvider::from_system_time()` ([src/lib.rs#L168-L199](#), [service.rs#L633-L649](#)).

The promised minimum cancellation window is therefore shorter than the future drift consensus accepts, and the clock that decides eligibility is a value the block producer chooses. Pallet hooks also run before a block's invariants and signed extrinsics, so Scheduler reads the timestamp inherited from the preceding block and a cancellation included in the same block cannot race it.

A miner who wins one QPoW block publishes a timestamp 29,900 ms ahead of wall-clock time. In the next block Scheduler's `on_initialize` services the bucket holding the transfer, pays the recipient and removes the entry from `PendingTransfers` before any signed extrinsic runs. The sender's `cancel`, dispatched 200 ms after scheduling and far inside the 12-second window, fails with `PendingTxNotFound`. The proof of concept runs the identical transfer, amount, delay and cancellation twice against the production runtime configuration: with honest timestamps the recipient's gain is 0, and with the accepted future timestamp it is the full amount, 10 times the existential deposit. Two executions of the same authorisation end with different owners of the funds, so this is direct loss of funds — the sender exercises a revocation right the protocol enforces, and the value is gone anyway. The attack needs one won block, not majority hash power, a privileged address or leaked credentials.

Proof of Concept

1. In block 1, with the on-chain timestamp at 35,000 ms, the sender calls `schedule_transfer_with_delay` for the production minimum delay of 12,000 ms; the target is 47,000 ms and the task is placed in the 72,000 ms bucket.
2. `PendingTransfers` contains the resulting transaction id.
3. In block 2 the attacker, having won the block, sets the timestamp to 65,000 ms while the importing node's wall clock reads 35,100 ms; `check_inherent` accepts it at 29,900 ms of drift.
4. In block 3, 200 ms of wall-clock time after scheduling, Scheduler's `on_initialize` reads the inherited 65,000 ms timestamp, normalises it to the 72,000 ms bucket, services that agenda and pays the recipient.
5. The sender's `cancel` in that same block fails with `PendingTxNotFound`, and the transaction id is gone from `PendingTransfers`.
6. Control: the identical run with timestamps advancing at wall-clock rate leaves the transfer pending, `cancel` succeeds, and the recipient's balance is unchanged.

IMM-BC-CRITICAL-88609

High-security gate inspects only the call, so an unbounded transaction tip drains a protected account in one block

Report ID	#88609
Severity	Critical
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield

Description

High-security mode exists to make an account's funds unstealable by a keyholder. The pallet README states that "an attacker who compromises account keys cannot disable protections to steal funds immediately" and that "attackers cannot bypass the delay period and guardian oversight" ([README.md#L69](#), [README.md#L91](#)). The primitives crate names the three controls — call whitelisting, a guardian able to cancel, and delayed execution — and describes the whitelist as covering "only operations that are reversible or delayed" ([src/lib.rs#L1-L9](#), [src/lib.rs#L82-L86](#)).

The gate that enforces this reads the call and nothing else:

```
fn validate(
    &self,
    origin: sp_runtime::traits::DispatchOriginOf<RuntimeCall>,
    call: &RuntimeCall,
    _info: &sp_runtime::traits::DispatchInfoOf<RuntimeCall>,
    _len: usize,
    _self_implicit: Self::Implicit,
    _inherited_implication: &impl sp_runtime::traits::Implication,
    _source: frame_support::pallet_prelude::TransactionSource,
) -> sp_runtime::traits::ValidateResult<Self::Val, RuntimeCall> {
    ...
}
```

— [transaction_extensions.rs#L60-L81](#), which then calls `HighSecurityConfig::is_call_allowed(&who, call)` and rejects with `InvalidTransaction::Custom(1)`. The tip is not part of the `RuntimeCall`. It is a field of `ChargeTransactionPayment`, a later member of the same extension tuple ([src/lib.rs#L154-L176](#)):

index	extension
6	<code>frame_system::CheckWeight</code>
7	<code>ReversibleTransactionExtension</code> — the high-security gate
8	<code>WormholeProofRecorderExtension</code>

index	extension
9	<code>'ChargeTransactionPayment'</code> — withdraws fee plus tip

The gate runs at index 7 and receives `_info` and `_len` without using either, so the only input it weighs is the call; `ChargeTransactionPayment::prepare` then withdraws the inclusion fee plus the tip from the same signer two positions later. The tip is therefore invisible to the policy, uncapped, withdrawn immediately, and corrected only for weight on post-dispatch, never refunded when the dispatch fails. The value does not burn: the fee sink is `pallet_mining_rewards::TransactionFeesCollector` ([configs/mod.rs#L477-L480](#)), and `on_finalize` mints the collected fees to the block author ([src/lib.rs#L192-L196](#)), who names himself with an unsigned 32-byte preimage in the pre-runtime digest ([src/lib.rs#L204-L208](#)). QPoW authorship is permissionless.

A keyholder attaches an arbitrary tip to `ReversibleTransfers::cancel { tx_id: H256::default() }` — whitelisted, and blessed as safe by the runtime's own test suite ([transaction_extensions.rs#L648-L654](#)) — guaranteed to fail and so carrying nothing but the tip. Measured on the frozen tree, 999,987,902,213,000 of 999,993,777,213,000 planck, 99.9994%, leaves the account in one block, and the residual is 5,875,000,000 planck. No `PendingTransfer` is created, so the guardian's `cancel` has no target and `recover_funds` ([src/lib.rs#L530-L545](#)) sweeps only that residual. When the attacker authors the block, his own balance gains 1,000,686,541,556,534 planck, paired with a wormhole tree leaf that makes the proceeds immediately spendable ([src/lib.rs#L228-L237](#)). With no authorship digest the same drain accrues to the treasury instead and the victim's residual is identical, so who collects the value is the attacker's choice and the victim's loss is not. All three advertised controls fail in the same block, and the delay window never opens at all: direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield.

Proof of Concept

1. The victim enrolls through a real signed `set_high_security` extrinsic naming a guardian and a 100-block delay; `is_high_security_account` returns true.
2. Control: a signed `Balances::transfer_keep_alive` from the same account is rejected by the gate and moves nothing.
3. The attacker signs `ReversibleTransfers::cancel { tx_id: H256::default() }`, a whitelisted call that is guaranteed to fail, with a tip sized to leave the account at the existential deposit.
4. `Executive::apply_extrinsic` accepts the transaction, the inner dispatch fails, and 999,987,902,213,000 planck — 99.9994% of the protected balance — has left the account.
5. `PendingTransfersBySender` for the victim is empty, the guardian's `cancel` errors for want of a target, and `recover_funds` returns only 5,875,000,000 planck.
6. With the attacker's own 32-byte preimage in the pre-runtime digest, `on_finalize` credits the attacker 1,000,686,541,556,534 planck, records a wormhole leaf for that credit, and emits `MinerRewarded` naming the attacker's address.

IMM-BC-CRITICAL-89819

Missing FIPS 204 domain separation lets a victim's message signature authorize a transaction

Report ID	#89819
Severity	Critical
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield

Description

Quantus implements ML-DSA from scratch and uses it as the runtime signature scheme, so `Verify for DilithiumSignatureScheme` is what decides whether an extrinsic is authorised by the account it claims ([traits.rs#L152-L185](https://github.com/immunefi-team/audit-comp-quantus-chain/blob/master/traits.rs#L152-L185)). FIPS 204 provides `ctx` as a first-class domain-separation input for exactly this purpose: a signature produced in one context must not verify in another.

Both of the integration's paths pass no context. `Pair::sign` takes an arbitrary `message: &[u8]` and signs it with `secret.sign(message, None, None)` ([scheme_macro.rs#L358-L371](https://github.com/immunefi-team/audit-comp-quantus-chain/blob/master/scheme_macro.rs#L358-L371)), and the verifier the runtime reaches checks it with `pk.verify(msg, sig, None)` ([scheme_macro.rs#L493-L501](https://github.com/immunefi-team/audit-comp-quantus-chain/blob/master/scheme_macro.rs#L493-L501)):

```
pub fn $verify_fn(pub_key: &[u8], msg: &[u8], sig: &[u8]) -> bool {
    match qp_rusty_crystals_dilithium::$module::PublicKey::from_bytes(pub_key) {
        Ok(pk) => pk.verify(msg, sig, None),
        Err(e) => {
            log::warn!("public key failed to deserialize {:?}", e);
            false
        },
    }
}
```

The `Verify` adapter first checks that the account matches the public key carried in the signature, deriving it as `hash_bytes(pubkey)` ([traits.rs#L191-L201](https://github.com/immunefi-team/audit-comp-quantus-chain/blob/master/traits.rs#L191-L201)), and then calls that context-free verification over the extrinsic's payload.

Because the context is empty on both sides, nothing distinguishes a signature over an arbitrary byte string from a signature over a SCALE-encoded `SignedPayload`. The same `Pair::sign` is the single signing primitive behind transaction construction and generic message signing alike — the Quantus SDK's `sign_message` signs caller-supplied bytes through it under the same empty context, and no `<Bytes>` wrapping exists to

keep the two apart. The account binding does not help: a raw-message signature carries the victim's real public key, so the binding succeeds before the context-free verification does.

An attacker builds a transfer from the victim to himself with the victim's correct nonce, era, fee, genesis hash and spec and transaction versions, and SCALE-encodes the `SignedPayload` — 119 bytes in the proof of concept, below the 256-byte threshold above which Substrate hashes the payload, so it is signed verbatim. He presents those opaque bytes to the victim as a message to sign, for instance as a login or ownership challenge; the victim is shown no transaction and discloses no key material. The returned signature, inserted unchanged into the prebuilt extrinsic, is accepted by the runtime, which attributes the extrinsic to the victim, dispatches the transfer, charges the victim the fee and increments the victim's nonce. The proof of concept moves exactly 100 QUAN, 100,000,000,000,000 planck, to the attacker, debits the victim 100,012,953,787,000 planck including the fee, and advances the victim's nonce to 1, with no access to the victim's secret key, for both shipped parameter sets. This is direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield.

Proof of Concept

1. The attacker builds a `Balances::transfer_keep_alive` from the victim to himself with the victim's correct nonce, era, genesis hash and spec and transaction versions, and SCALE-encodes the `SignedPayload` — 119 bytes.
2. The attacker presents those bytes to the victim's key holder as an opaque message; the holder signs them through the generic `Pair::sign` operation and returns only the signature and public key.
3. The attacker inserts that signature unchanged into the prebuilt extrinsic and submits it through `Executive::apply_extrinsic`.
4. The runtime accepts the account-to-public-key binding, verifies the signature under `ctx = None`, and dispatches: 100,000,000,000,000 planck moves to the attacker, the victim is debited 100,012,953,787,000 planck including the fee, and the victim's nonce advances to 1.
5. Control: the identical payload signed under a non-empty FIPS 204 context is rejected by the shipped verifier, changing no balance and no nonce.
6. Control: with a non-empty context bound into the verify call, the unchanged exploit fails with `InvalidTransaction::BadProof`; restoring the empty context restores the transfer.
7. Both executed runs — one per shipped parameter set, Dilithium65 and Dilithium87 — produce the same outcome.

IMM-BC-HIGH-89065

Wormhole deposits below the circuit's fee-exitable minimum are recorded but can never be exited, freezing user funds

Report ID	#89065
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of funds

Description

Quantus holds value on keyless wormhole accounts, whose Poseidon addresses have no signing key, so value leaves them only through a ZK proof against `pallet-zk-tree`. The runtime floors keyless vesting payouts at one QUAN because a sub-quantum transfer "would be committed as a zero-value leaf, stranding funds paid to keyless beneficiaries" ([configs/mod.rs#L562-L572](#)).

`balances.transfer_keep_alive` is the only deposit path, and `WormholeProofRecorderExtension` records the raw planck of every native transfer except an exactly zero credit ([src/lib.rs#L1354-L1369](#)). `hash_leaf` commits `floor(amount / 1010)` ([tree.rs#L146-L148](#)), and the circuit's fee relation bounds that quantum from below, as the pallet mirrors it: "the smallest valid exit (one quantum out forces `input ≥ 2` quanta)" ([src/lib.rs#L1066-L1082](#)).

`Balances` nevertheless accepts, and the recorder records, credits far below that floor. `EXISTENTIAL_DEPOSIT` is `MILLI_UNIT = 109` planck ([src/lib.rs#L97-L102](#)), a tenth of one leaf quantum and a twentieth of the fee-exitable minimum: a one-quantum deposit quantizes to 1 and admits no positive output, and one at the existential deposit quantizes to 0. Vesting was gated for this hazard; permissionless `balances.transfer_keep_alive` was not.

Any credit in `[109, 2×1010)` planck to a keyless account is therefore frozen in full. The destination cannot sign it away, since no ML-DSA-65 key hashes to a Poseidon address, and cannot exit it: no positive output satisfies the range-checked fee constraint, and `process_exit_bundle` skips every slot whose `summed_output_amount` is zero ([src/lib.rs#L787-L789](#)). Leaves do not merge — the PoC sends ten one-quantum transfers, stranding 0.10 QUAN, five times the fee-exitable floor, as ten unexitable leaves, while the same 0.10 QUAN sent once exits 0.09 QUAN. The result is permanent freezing of funds.

Severity note. The stranded balances remain recoverable by the chain's own governance: a runtime upgrade can raise the existential deposit, reject sub-floor credits, or `force_transfer` the affected accounts, so the freeze does not require a hardfork to undo.

IMM-BC-HIGH-88969

Multisig re-decodes opaque call bytes without a depth limit, letting free stack-exhaustion poison consume block authors

Report ID	#88969
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Causing network processing nodes to process transactions from the mempool beyond set parameters

Description

FRAME's Executive decodes every incoming extrinsic under the structural depth bound `MAX_EXTRINSIC_DEPTH = 256` ([frame/support/src/lib.rs#L40](#)) on both the `apply_extrinsic` and `validate_transaction` paths ([src/lib.rs#L877-L881](#)), which keeps an attacker-supplied SCALE value from driving the codec into unbounded recursion.

`pallet-multisig` stores a proposal's inner call as opaque bytes, `BoundedCallOf<T> = BoundedVec<u8, MaxCallSize>` ([src/lib.rs#L251](#)), bounded by encoded byte length alone, with `MaxCallSize = 10240` ([configs/mod.rs#L648](#)). `Multisig::propose` turns those bytes back into a call with a plain recursive `RuntimeCall::decode` whose own comment records the cost as structural: "Decode cost is $O(\text{inner_call_count})$ for nested calls, not $O(\text{bytes})$ " ([src/lib.rs#L597-L600](#)).

Executive's depth limit governs the outer extrinsic and does not recurse into a `BoundedVec<u8>`'s contents, so the inner decode is a second, unprotected trust boundary. Transaction-queue validation never dispatches `propose`, so nothing reaches that decode before block construction. A 10,239-byte payload of 3,412 nested `Utility::batch` wrappers is canonical SCALE that fits inside the byte bound and exhausts the runtime's stack while decoding.

The production `WasmExecutor` contains the resulting out-of-bounds access as a trap, which `sc-block-builder` surfaces as a runtime-API error and rolls back ([service.rs#L556-L560](#)): the PoC sees the tentative fee, the nonce increment and 1,000,906,093,201 units of `ref_time` all restored and the extrinsic revalidating, so the work is free and repeatable. Basic authorship counts only validity errors reporting `exhausted_resources()` against its eight-skip allowance, so this generic runtime-API error keeps the author in its selection loop past the soft deadline. The PoC measures 340 apply/trap/rollback cycles, from 340 independently signed poisons inside the default pool limits, at 18.95 seconds — more than twice the roughly 9-second selection budget implied by the 10-second proposal duration QPoW passes to the proposer

([service.rs#L489-L499](#)). Each default author spends its whole selection phase on candidates that contribute no extrinsic, weight, fee or nonce to the block, causing network processing nodes to process transactions from the mempool beyond set parameters.

IMM-BC-HIGH-88250

Transient accept errors terminate the litep2p listener permanently, letting unauthenticated TCP floods remove a node from the network

Report ID	#88250
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Network not being able to confirm new transactions (total network shutdown)

Description

Quantus nodes network over a vendored litep2p stack whose transport manager is meant to cap inbound load at `MAX_CONNECTIONS_ESTABLISHED_INCOMING = 10_000` ([litep2p/mod.rs#L520-L522](#)). The manager enforces that cap by comparing `incoming_connections.len()` against the configured maximum ([limits.rs#L104-L113](#)), a set populated only by `accept_established_connection` ([limits.rs#L144-L156](#)), so a connection that has not completed the noise handshake is never counted — and nothing inside the node bounds the pre-handshake population, the repository shipping no `setrlimit` call, no `LimitNOFILE` unit and no container limit.

When the descriptor table saturates, `accept` returns `EMFILE`. That error is transient — the descriptors free within milliseconds — but the TCP accept loop returns `Poll::Ready(None)` on any error, ending the `Stream`: no retry, no backoff, no re-listen ([tcp/mod.rs#L592-L600](#)). The WebSocket transport carries the identical arm ([websocket/mod.rs#L634-L642](#)).

A remote party that completes no handshake, holds no key material and sends no payload opens roughly a thousand plain TCP connections in about 60 milliseconds and exhausts a node launched with the common 1024-descriptor limit. The listener terminates, and in the same second the transports, the litep2p backend and the `block-announces` and `light` protocol services all close, so the node can no longer receive blocks, announce its own, or acquire peers. The PoC reproduces this 4/4, including at 4,065 connections against a 4,096-descriptor limit — raising the limit prices the attack rather than preventing it. The attacker then disconnects; descriptors return to baseline and the process stays alive, yet the honest peer that connected before the burst can never reach it again. Only an operator restart restores networking, after which the node can be killed again immediately. Applied across the reachable node set, including the publicly addressable bootnodes, this leaves the network not being able to confirm new transactions.

Severity note. Only the per-node shutdown is demonstrated; the extension to the whole network is reasoned rather than measured, since reproducing it needs a live network of independently operated nodes that the competition scope does not provide.

IMM-BC-HIGH-88404

Scheduler drops the caller's preimage reference on a failed enactment retry, silently voiding a passed referendum

Report ID	#88404
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results

Description

`pallet-referenda` owns exactly one preimage reference for a `Lookup` proposal and releases it exactly once. `submit` requests the preimage so the proposal's bytes survive a noter reclaiming their deposit, on the stated guarantee that "on approval the scheduler holds its own request from `schedule_named` until dispatch, so the bytes stay pinned through enactment" ([src/lib.rs#L574-L597](#)); that single reference is released by `conclude_ongoing`, whose lone `T::Preimages::drop(proposal)` runs on every terminal transition ([src/lib.rs#L1705-L1707](#)).

The scheduler's half of that split is its own reference: `schedule_named` routes to `do_schedule_named_inner` with `request_preimage = true`, and on success adds a reference under the flag, leaving `referenda`'s intact ([src/lib.rs#L962-L965](#)). `Referenda` places the enactment task through a 17-attempt loop that slides one block forward whenever the target agenda is full (`MAX_ENACTMENT_SCHEDULE_RETRIES = 16`, [src/lib.rs#L1077](#)).

Every fallible path in `do_schedule_named_inner` drops a reference unconditionally, ignoring `request_preimage` — including the `place_task` failure a full agenda produces ([src/lib.rs#L955-L961](#)). The scheduler holds no reference of its own at that point, so the drop consumes the caller's.

A single failed first attempt therefore leaves an approved proposal completely unpinned.

`RequestStatusFor[hash]` reads `Requested { count: 1 }` after `submit`, `Unrequested` after the failed attempt, `Requested { count: 1 }` again once the retry succeeds — that one belonging to the scheduler — and `Unrequested` when `conclude_ongoing` drops what it believes is the submit-time reference. The noter's ordinary, permissionless `unnote_preimage` then deletes the bytes. At the scheduled block the scheduler emits `CallUnavailable`; the referendum still reads `Approved` and retains neither call nor origin, so no on-chain retry exists and recovery needs a full re-proposal cycle. A full agenda is reachable:

`MaxScheduledPerBlock = 50` ([configs/mod.rs#L345](#)), low-priority tasks saturate at 40 of those, and every vote

re-arms a priority-128 referendum alarm at `now + 1`, so votes cast in the preceding block close the gap. The outcome is manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results.

Severity note. The on-demand trigger the Critical framing rests on does not hold, so a member cannot aim the failure at a chosen referendum; the defect remains reachable through the accidental route — a full enactment-block agenda when scheduling runs, plus an honest `unnote_preimage` — which is what the final severity reflects.

IMM-BC-HIGH-88938

Wormhole miner volume-fee credit records no ZK-tree leaf, stranding the keyless block author's share

Report ID	#88938
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of funds

Description

A Quantus block author is a keyless wormhole account: `extract_author_from_digest` returns `derive_wormhole_address(preimage)` of the miner-supplied pre-runtime payload, never the raw bytes ([src/lib.rs#L143-L148](#)), while a signable `AccountId` is `hash_bytes(ML-DSA public key)` over a disjoint encoding ([traits.rs#L194-L200](#)). The account can never sign, so its only spend instrument is a ZK-tree leaf redeemed through a wormhole exit — which is why `pallet-mining-rewards` records a transfer proof on every mint to that same payee ([src/lib.rs#L230-L237](#)).

`process_exit_bundle` splits the wormhole volume fee — `VolumeFeesBurnRate` is 50% ([configs/mod.rs#L734-L736](#)) — and mints the remainder to that author through `increase_balance`, whose `Ok` arm carries no `record_transfer` ([src/lib.rs#L955-L965](#)), while the exit credits earlier in the same function, using the same primitive, record theirs explicitly ([src/lib.rs#L851-L859](#)). No fallback recorder covers the miner credit: `increase_balance` is event-free by deliberate design here, so the runtime's `WormholeProofRecorderExtension`, which matches only `Balances::Transfer`, `Minted`, `TransferOnHold` and `ReserveRepatriated` ([transaction_extensions.rs#L339-L372](#)), has nothing to match, and the scan cannot run for this call in any case, because `verify_private_batch` is admitted unsigned through the pallet's own `validate_unsigned` ([src/lib.rs#L1002-L1019](#)) and the extension implements no bare post-dispatch hook.

Every miner volume-fee share is therefore credited to an account that can neither sign nor prove. The author's `TransferCount` stays at zero, no leaf binds the amount, and the balance is unreachable for good — on every block that settles a wormhole exit, with no attacker and no privileged actor on the path. A leaf commits `floor(amount / 1010)` quanta ([tree.rs#L146-L148](#)), so the value a recorded leaf would have made spendable is `floor(share / 1010) * 1010`. The production harness measures 20,000,000,000 planck per settling block at 100 UNIT of exit volume and 200,000,000,000 planck at 1,000 UNIT, which at the 2,628,000 blocks per year implied by `TARGET_BLOCK_TIME_MS = 12_000` ([src/lib.rs#L87](#)) is 52,560 and 525,600 UNIT per year respectively. This is permanent freezing of funds.

Severity note. The quantization bounds what is at stake: at the fee floor and at exit volumes of 1 and 10 UNIT per block the miner share is half a quantum, so a recorded leaf would have committed zero and nothing would have been recoverable; the loss becomes non-zero only once per-block exit volume reaches roughly 100 UNIT.

IMM-BC-HIGH-88498

Mining rewards are minted at planck precision while the miner's only leaf commits whole quanta, stranding the remainder

Report ID	#88498
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of unclaimed yield

Description

Quantus pays mining rewards to a wormhole-derived block-author address, `H(H(salt || secret))`, which has no ML-DSA private key matching its 32-byte account identifier; a miner redeems that value only by proving ownership of a recorded wormhole leaf. The chain states the resulting invariant where it governs its other keyless payee: `pallet-vesting` rounds every payout down to a multiple of the leaf quantum because "ZK-tree leaves commit `amount / quantum`, so a payout below one quantum would create a zero-value leaf: funds moved to a keyless beneficiary would be irrecoverable" ([src/lib.rs#L147-L152](#)), and the runtime anchors that quantum to the wormhole constant ([configs/mod.rs#L562-L572](#)).

`MiningRewards::on_finalize` computes the block reward from the remaining supply and splits it between treasury and miner at raw `u128` precision ([src/lib.rs#L155-L162](#)), then settles accumulated transaction fees and the block reward through two separate `mint_reward` calls ([src/lib.rs#L193-L196](#)), each of which mints the exact raw amount and records a transfer proof for that same raw amount ([src/lib.rs#L230-L237](#)).

The leaf does not preserve that precision. `hash_leaf` commits `floor(amount / AMOUNT_SCALE_DOWN_FACTOR)` ([tree.rs#L146-L148](#)) with the factor fixed at `10_000_000_000` ([tree.rs#L34](#)), so a commitment resolves only to 0.01-QUAN increments, and neither reward path aligns the miner's share to that quantum or carries a remainder forward.

Each payout's sub-quantum remainder therefore sits in the miner's raw free balance and in the `MinerRewarded` event, but is absent from the only commitment the exit circuit can consume, and because every payout takes its own transfer count and nullifier, a later reward cannot consolidate or recover an earlier leaf's omitted remainder. The first block credits 689,151,854,809 planck while its leaf commits 680,000,000,000, leaving 9,151,854,809 planck unclaimable. Projected over the first year at `EmissionDivisor = 15_163_560` ([configs/mod.rs#L144](#)) and the 12-second target block time, the PoC gives 2,628,000 rewarded blocks, 1,662,837.68 QUAN of miner block rewards and 13,083.01 QUAN omitted from leaf commitments — 0.786788% of miner block-reward yield. Transaction-fee rewards are affected

independently: the full-runtime PoC charges an ordinary signed transfer a fee of 10,995,787,000 planck, credits that exact fee to the miner, and shows the fee leaf committing only 10,000,000,000 planck. This is permanent freezing of unclaimed yield.

IMM-BC-HIGH-88485

Wormhole pool deduplication keys on unverified nullifiers, letting one proof become many fee-free mempool entries

Report ID	#88485
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Causing network processing nodes to process transactions from the mempool beyond set parameters

Description

Wormhole exit transactions are unsigned, fee-free and propagated through the transaction pool. Quantus deliberately moves the expensive Plonky2 verification out of pool admission and into the block-inclusion gate, as `pre_dispatch` records: this "runs the FULL validation including ZK verification ... what keeps unverified/junk proofs out of blocks now that pool admission is verify-free" ([src/lib.rs#L1045-L1055](#)). Because proofs are unverified at admission, a second mechanism carries the load: different byte representations of one logical exit share a `provides` tag and so occupy a single pool identity.

That tag is the blake2-256 of the bundle's nullifiers ([src/lib.rs#L1247-L1254](#)), attached by `validate_unsigned` with a constant priority, longevity 5 and `propagate(true)` ([src/lib.rs#L1021-L1026](#)). The cheap pre-validation that runs first checks the nullifiers for intra-segment uniqueness and for absence from `UsedNullifiers` and the bundle's already-claimed set ([src/lib.rs#L695-L701](#)).

Those checks establish only that the attacker-declared nullifier is fresh. They do not establish that it is the nullifier this proof commits to — the deferred Plonky2 verification is what establishes that. The value that decides whether two fee-free submissions are duplicates is therefore attacker-controlled at the moment it is trusted.

One existing serialized proof, with a single nullifier public input changed and re-serialized, stays canonically encoded, passes every cheap check, receives a different `provides` tag, and enters the pool as an independent, propagatable transaction requiring no account, signature, balance or fee. The regression PoC turns one unchanged cryptographic proof into 32 independently admissible pool identities, each rejected only at `pre_dispatch`, after full verification has run. Every variant therefore costs a processing node a complete private-batch verification before it is discarded — benchmarked at 15.162 ms ([weights.rs#L74-L75](#)). At roughly 157.5 KB per proof, 560 such variants occupy about 88 MB, well inside the 262,144 KiB default pool byte limit ([transaction_pool_params.rs#L51-L55](#)), and carry about 8.49 seconds of verification against

the 10-second proposal duration the node passes to the proposer ([service.rs#L489-L499](#)). Generating them needs no new valid proof, causing network processing nodes to process transactions from the mempool beyond set parameters.

IMM-BC-HIGH-89062

Unsigned wormhole exit proofs are verified without limit per block, letting any peer exhaust every producer's build budget

Report ID	#89062
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Temporary freezing of network transactions by delaying one block by 500% or more of the average block time of the preceding 24 hours beyond standard difficulty adjustments

Description

The wormhole settles private-batch exits through `verify_private_batch`, a bare unsigned extrinsic that calls `ensure_none(origin)` and returns `Pays::No` ([src/lib.rs#L583-L611](#)), so no account, balance or signature is needed to submit one. The pallet splits validation in two: pool admission runs only cheap pre-validation, because a full verify there "would let unsigned, fee-free traffic force unbounded verification work per gossiped proof" ([src/lib.rs#L1006-L1011](#)); the expensive half moves to `pre_dispatch`, the block-inclusion gate, which runs the ZK verification and returns `InvalidTransaction::Call` for a proof that fails ([src/lib.rs#L1045-L1062](#)).

However, nothing bounds how many failed verifications a producer performs while building one block. Returning `Err` only excludes the candidate: the rejection consumes no block weight and pays no fee. Over the node's ten-second build window ([service.rs#L489-L499](#)) the producer walks the ready queue, verifying one candidate after another at 15.16 ms each ([weights.rs#L67-L75](#)), and a rebuild on each arrival, up to one per two seconds ([src/lib.rs#L474](#)), makes it walk the queue again.

Any peer reaches this with the committed 157,500-byte private-batch fixture ([private_batch.hex](#)), retargeted at a recent block with randomized nullifiers: the genuine proof body passes every admission check, the fresh nullifiers defeat pool dedup, and only the final public-input check fails after the whole verification has run. Over a three-minute run the producer performed 45,744 full verifications, admitted zero junk into any block, and its build time rose from 0–2 ms to a median of 2,837 ms and a maximum of 7,668 ms; block intervals stretched to a mean of 43.4 s and a maximum of 142.4 s against the runtime's 12,000 ms target ([src/lib.rs#L87](#)), and signed traffic confirmed at 81.3 s mean latency against 21.8 s at rest. One anonymous peer sustaining roughly 8 MB/s holds every producer there, which is temporary freezing of

network transactions by delaying one block by 500% or more of the average block time of the preceding 24 hours beyond standard difficulty adjustments.

IMM-BC-HIGH-89247

Wormhole inertness check keys on nullifiers instead of value, letting zero-value exits settle free and grow state

Report ID	#89247
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Inaccuracies to the weight (similar to gas) accounting system that under or overcharge by more than 50% of the correct amount (and are not intentional as indicated by the program's terms)

Description

Wormhole exit settlement is unsigned and `Pays: :No`, gated economically only by the volume fee, and its only emptiness check reads nullifiers, not value: a segment is inert only when every nullifier is zero ([src/lib.rs#L652-L654](#)), and only an all-inert bundle is rejected ([lib.rs#L721-L722](#)). The circuit gives each dummy leaf a nullifier of `H(H(preimage))`, so padding is indistinguishable from real slots ([wormhole-zk.md#L60-L61](#)).

A private-batch segment is therefore never inert, and nothing else gates value ([lib.rs#L1095-L1114](#)): `volume_fee_for_exit` derives the fee from minted value alone ([lib.rs#L1082-L1092](#)), and the success path returns `Pays: :No` with no weight refund ([lib.rs#L998](#)), so a settlement that mints nothing pays nothing.

`verify_public_batch` is priced at a fixed worst case of 742 exit slots ([weights.rs#L87-L98](#)) regardless of content ([weights.rs#L184-L196](#)): 3,744 ms against the 4,500 ms Normal budget ([configs/mod.rs#L72-L88](#)), 83.2% of a block, charged in full for a settlement that mints nothing. Each settlement also writes a permanent `UsedNullifiers` entry per leaf slot ([lib.rs#L764-L774](#)), never pruned: at 52 private-batch settlements per block that is 366 entries a block, about 264 MB per day of consensus state every node carries forever. This is an inaccuracy to the weight accounting system that overcharges by more than 50% of the correct amount.

Severity note. The block-stuffing component is bounded: `UNSIGNED_EXIT_PRIORITY` is the constant 1 ([lib.rs#L28](#)), so these settlements cannot outbid fee-paying traffic and fill only genuinely idle capacity. Priority does not bound the other two components: it decides who wins contested capacity, not what a settlement writes once included, so the never-pruned state growth and the fixed worst-case weight charged to an inert settlement both stand at full effect, and that is what carries the finding above the Medium band its individual impact strings occupy.

IMM-BC-HIGH-88566

Bundle-level fee rounding settles less than the per-leaf fee the circuit locks, stranding the difference

Report ID	#88566
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of unclaimed yield

Description

The wormhole leaf circuit enforces its fee relation independently for every leaf: $(out_1 + out_2) \cdot 10000 \leq input \cdot (10000 - fee_bps)$ ([wormhole-zk.md#L29-L32](#)). The runtime is meant to settle exactly the minimum those proofs lock. `volume_fee_for_exit` states that invariant in its own doc comment: it "[m]irrors the circuit's integer fee relation over QUANTIZED amounts", and "[c]omputed over the bundle's minted total, the result never exceeds what the individual proofs locked (ceil of a sum $\leq$ sum of ceils)" ([src/lib.rs#L1066-L1081](#)).

`process_exit_bundle` accumulates every successfully minted exit into `minted_exit_amount` ([lib.rs#L849](#)) and then applies that single ceiling once, to the aggregate ([lib.rs#L867-L872](#)), splitting the result between the burn, the miner and the public-batch aggregator.

However, "never exceeds" is not "equals", and the inequality is strict for small leaves. At the production rate of 4 bps ([configs/mod.rs#L734](#)) a leaf holding a 2-quantum input can output at most 1 quantum, because $2 \cdot 10000 \leq 2 \cdot 9996$ is false, so the circuit forces a full one-quantum gap on that leaf. A private batch carries seven leaves ([build.rs#L39-L44](#)), so seven such leaves lock 7 quanta of fee while the runtime settles $\text{ceil}(7 \cdot 4 / 9996)$, one quantum.

The remaining 6 quanta — 0.06 QUAN — go nowhere. They are not returned as an exit output, not burned, not credited to the miner and not credited to the aggregator, and they are not recoverable: all seven nullifiers are written to `UsedNullifiers` in the same call ([lib.rs#L764-L774](#)), so the leaves cannot be spent again. The end-to-end proof of concept builds seven real leaf proofs, aggregates them, passes `validate_unsigned` and `pre_dispatch`, settles through `verify_private_batch`, and measures a total issuance delta of exactly one quantum against a proof-enforced 7, with all seven nullifiers consumed. The same settlement path serves `verify_public_batch`, which at the production size of 53 inner batches ([build.rs#L46-L52](#)) carries 371 leaves: 371 quanta locked, $\text{ceil}(371 \cdot 4 / 9996) = 1$ quantum settled, 370

quanta or 3.70 QUAN stranded per batch. That value is fee yield the burn, miner and aggregator legs were owed and will never receive, which is permanent freezing of unclaimed yield.

IMM-BC-HIGH-88223

Wormhole miner fee and aggregator rebate are credited without a zk-tree leaf, freezing them permanently

Report ID	#88223
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of funds (fix requires hardfork)

Description

The block author's share of the wormhole volume fee is credited to an address derived from the miner-supplied pre-runtime preimage: `extract_author_from_digest` rehashes that preimage through `derive_wormhole_address` ([src/lib.rs#L143-L148](#)), documented as `address = hash(hash(salt + secret))` ([src/lib.rs#L97-L111](#)). A signable account is instead Poseidon over the ML-DSA public key ([scheme_macro.rs#L239-L245](#)), so no keypair produces a `rehash_to_bytes` image and the author address is spendable only through a wormhole ZK exit. An exit is bounded by a zk-tree leaf, and `record_transfer` is the only thing that creates one; it bumps the recipient's transfer count and inserts the leaf ([src/lib.rs#L1294-L1310](#)), which is why the design records each mint "so the new mint becomes a fresh leaf available for future wormhole exits" ([wormhole-zk.md#L88-L90](#)).

Every other credit path follows that pattern. The exit loop calls `Self::record_transfer` immediately after a successful `increase_balance` ([lib.rs#L828-L859](#)), and `pallet-mining-rewards`, which credits the same class of address, calls `record_transfer_proof` right after `mint_into` ([src/lib.rs#L230-L237](#)).

The miner-fee credit does not:

It deposits an event and stops. The public-batch aggregator rebate has the identical omission ([lib.rs#L919-L933](#)), and `record_transfer` is reached from only four sites in the pallet — `lib.rs:486`, `lib.rs:854` inside the exit loop, and `lib.rs:1367` and `lib.rs:1386` inside the `record_transfer_proof` trait implementation — so nothing sweeps or backfills these credits.

The miner therefore receives free balance at an address it can neither sign for nor prove, and the value is frozen the moment it is credited. It fires on every successful exit that produces a non-zero miner fee, with no attacker involved: at `VolumeFeeRateBps = 4` and `VolumeFeesBurnRate` of 50% ([configs/mod.rs#L730-L738](#)) the miner share is 2 bps of all wormhole exit volume, accruing forever and growing with wormhole adoption. The aggregator leg freezes the same way. The regression test added with the earlier fix asserts only that

the balance arrived ([tests.rs#L1088-L1091](#)), never that a leaf exists, and the proof of concept turns that into an on-chain witness: after `verify_private_batch` the author's free balance has strictly increased while `TransferCount[author]` is still `0` and `ZkTree::leaf_count()` grew by exactly the number of exit accounts. Recovery requires a runtime upgrade that re-issues the credits, which is permanent freezing of funds (fix requires hardfork).

Severity note. The frozen value is protocol fee yield rather than user principal — no deposit or exit output is touched, and the per-exit amount is the 2 bps miner share — which is what holds the finding below Critical. What keeps it at High is that the loss is deterministic on every exit, has no in-protocol recovery path at all, and accrues without bound.

IMM-BC-HIGH-89139

Credits to wormhole addresses are not quantized, so sub-quantum value is committed to no leaf and unexitable

Report ID	#89139
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Permanent freezing of funds

Description

Native balances carry twelve decimal places — `UNIT` is `1_000_000_000_000` ([src/lib.rs#L96-L102](#)) — while a zk-tree leaf commits amounts at two, quantized by `AMOUNT_SCALE_DOWN_FACTOR = 10_000_000_000` ([tree.rs#L34](#)), so one leaf quantum is 0.01 QUAN. A wormhole address has no signing key, so its balance moves only through a ZK exit, and that exit is bounded by the `input_amount` the leaf commits.

`WormholeProofRecorderExtension` scans balance events and passes every credit it finds to `record_transfer_proof` ([transaction_extensions.rs#L309-L396](#)). That recorder rejects exactly one thing, an exact zero ([src/lib.rs#L1354-L1356](#)); every other native credit reaches `record_transfer`, which stores the full-precision amount, advances the recipient's transfer count and inserts a leaf ([lib.rs#L1294-L1310](#)). `hash_leaf` then commits the amount:

However, nothing on that path requires the credited amount to be a multiple of the quantum. Floor division silently drops `amount mod 1010` from the Merkle commitment, and the circuit takes the quantized `u32` as its private `input_amount`, so no valid membership witness can ever cover the remainder. Storage keeps the full precision; the hash the exit is proved against does not.

For a keyless wormhole recipient the remainder is unreachable by any in-protocol path: it cannot be signed for and it cannot be proved. The existential deposit is one `MILLI_UNIT`, 0.001 QUAN ([src/lib.rs#L98-L102](#)), one tenth of a leaf quantum, so a credit of 0.009 QUAN is above the deposit and succeeds — the balance arrives, a leaf is inserted and the transfer count advances — while the leaf commits an input of zero and the entire credit becomes unexitable. Larger non-aligned credits lose their remainder the same way: 1.009 QUAN commits 1.00, and 10.005 QUAN commits 10.00. The circuit's fee relation compounds it at the bottom of the range: $(out_1 + out_2) \cdot 10000 \leq input \cdot (10000 - fee_bps)$ ([wormhole-zk.md#L29-L32](#)) at the configured 4 bps ([configs/mod.rs#L734](#)) caps a one-quantum input's output at zero. The failure is silent throughout: the transfer succeeds, the balance rises by the full amount, an event and a leaf are produced,

and nothing signals that the value cannot be withdrawn. Each sub-quantum credit also inserts a worthless leaf and advances the transfer count, growing the tree permanently. This is permanent freezing of funds.

Severity note. The loss per credit is bounded by one leaf quantum: any amount above 0.01 QUAN strands only $\text{amount} \bmod 10^{10}$ under 0.01 QUAN of value, which is what holds the finding below Critical. Credits between the 0.001 QUAN existential deposit and the 0.01 QUAN quantum lose 100% of their value, and every loss is permanent with no recovery path, which keeps it at High.

IMM-BC-HIGH-88297

Scheduled transfers below the existential deposit can never execute and only the guardian can clear them

Report ID	#88297
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Temporary freezing of funds for at least 24 hour

Description

A high-security account moves value only through `pallet-reversible-transfers`: the runtime's call whitelist admits `schedule_transfer`, `cancel`, `recover_funds` and `Vesting::claim`, and nothing else ([configs/mod.rs#L669-L693](#)). `schedule_transfer` places the amount on hold and hands the scheduler a named task to dispatch after the account's delay.

`do_schedule_transfer_inner` validates exactly one property of the amount:

At execution, `do_execute_transfer` builds a `pallet_balances::Call::transfer_keep_alive`, releases the hold, removes the pending metadata and the sender-index entry, dispatches, and returns the dispatch result as its own ([src/lib.rs#L693-L738](#)).

However, `transfer_keep_alive` will not create an account below the existential deposit, one `MILLI_UNIT` in production ([src/lib.rs#L98-L102](#)), and nothing between the schedule-time check and the dispatch tests for that. The dispatch is the last step and its failure becomes the extrinsic's failure, so the release and the metadata removal roll back — while the scheduler has already consumed the named task.

`do_schedule_transfer_inner` never calls `set_retry`; the pallet's only `Scheduler::Retries` references are benchmark annotations in its weights file. The hold is back, the pending entry is back, and nothing will ever execute it.

The owner cannot clear it either. `cancel_transfer` reads the guardian frozen into the pending record, and for a high-security schedule `pending.guardian != pending.from`, so the owner is rejected with `InvalidReverser` and only the guardian can cancel ([lib.rs#L865-L885](#)). When the guardian does, the released funds go to the guardian, minus a 1% volume fee that is burned ([lib.rs#L918-L954](#), [configs/mod.rs#L531](#)). `recover_funds` is guardian-gated by the same error and also pays out to the caller, the guardian ([lib.rs#L530-L568](#)). No extrinsic returns a stranded transfer to its sender.

Sending a small amount to a new wallet before moving a real balance is routine, and it is exactly the case that breaks. Each stranded transfer locks its own amount, bounded by the existential deposit, and permanently consumes one of the 16 slots in `MaxPendingPerAccount` ([configs/mod.rs#L529](#)). After 16, the account can schedule nothing further, and `schedule_transfer` is the only whitelisted call that sends its own balance outward. The proof of concept schedules `EXISTENTIAL_DEPOSIT / 2` to an unfunded address in the runtime integration harness, runs past the execution window and 500 blocks beyond it, and observes a recipient balance of zero with the pending entry still held, then confirms the owner's cancel fails and the guardian's succeeds. The end state is an account that cannot move its own money, holding funds it cannot reach, which is temporary freezing of funds for at least 24 hour.

IMM-BC-MED-88219

Invalid proof-of-work is not attributed to the sending peer, so a hostile peer is never penalised

Report ID	#88219
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

Quantus replaces upstream `sc-consensus-pow` with its own import pipeline, whose import queue is expected to charge a misbehaving peer for the blocks it supplies. The `Verifier` implementation the queue calls, `SimplePowVerifier::verify`, only pops the seal digest ([src/lib.rs#L442-L453](#)). The proof-of-work check moved into `import_block`, where it runs after `check_inherents` has instantiated WASM and executed a runtime call over the attacker's entire block ([src/lib.rs#L249-L262](#)), and the node passes `0` for the height after which inherents are checked, so no block escapes it.

Because the check no longer happens in the verifier, a failed seal surfaces as `BlockImportError::Other` rather than `VerificationFailed`. The adjacent match arms diverge completely: [chain_sync.rs#L747-L762](#) drops the peer with `rep::VERIFICATION_FAIL`, while the arm that actually receives invalid proof-of-work does not:

`restart()` additionally cancels in-flight requests to every connected peer, not only the offender. A peer therefore serves blocks with forged proof-of-work indefinitely at no cost. Over a 150-second two-node run the victim logged 18 rejections, 7 sync restarts and 9 peer re-admissions with zero reputation actions, and never advanced past genesis; in a three-node run its keep-up ratio against an honest miner fell from 100% to 79.7%. Each junk block costs roughly a kilobyte on the wire and buys a WASM runtime call plus a full sync reset — damage to the users and the protocol with no profit motive for the attacker.

IMM-BC-MED-88224

Unvalidated peer-advertised block height pauses mining and suppresses block-import notifications

Report ID	#88224
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

The block-announces handshake carries a peer's claimed `best_number`, which `ChainSync` records before any header or cumulative work is validated. `median_seen` returns the median of every peer's claimed height, taking index `len / 2` — the upper median for an even peer count ([chain_sync.rs#L1573-L1585](#)). When that median runs far enough ahead the node enters major-sync state, and the mining loop declines to propose while the sync oracle reports major syncing ([service.rs#L341-L349](#)); that check sits before the zero-peer guard, so `--force-authoring` does not bypass it. Major sync also relabels imported blocks `BlockOrigin::NetworkInitialSync` ([chain_sync.rs#L1618](#)), and that origin suppresses import notifications entirely:

Nothing requires the peer to deliver a header for the tip it claims, or to show any work, before the number counts toward the median. One peer advertising a height of 1,000,000 put a victim miner into  Syncing ... `target=#1000000`, produced 22 `Mining paused` events and cut production to 14 blocks in 120 seconds against a ~102 baseline. Against a non-mining node the same lie blackholed `chain_subscribeNewHeads` for 101.0 seconds: blocks 82–133 were imported but never announced, the node's own informant printed no `Imported` line for any of them, and `system_health` kept reporting normally. Recovery is unaided after roughly 111 seconds, and a fresh peer identity every ~38 seconds sustains the window — damage to the users and the protocol with no profit motive for the attacker.

IMM-BC-MED-88226

`multistream-select` sizes its read buffer to an unauthenticated peer's declared frame length

Report ID	#88226
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Increasing network processing node resource consumption by at least 30% without brute force actions, compared to the preceding 24 hours

Description

Protocol negotiation runs on a plaintext socket before any cryptography: `negotiate_connection` negotiates `/noise` and only then calls `noise::handshake` ([connection.rs#L434-L444](#)). Inside that negotiation the framing layer reads an unsigned-varint length prefix and immediately sizes its buffer to the value the remote peer declared:

`read_buffer` is a `BytesMut` and `BytesMut::resize` writes the fill byte, so the pages are touched and become resident rather than staying virtual. `MAX_LEN_BYTES = 2` caps `len` at 16,383 ([length_delimited.rs#L30-L31](#)), so no bounds check is missing.

However, nothing requires the peer to send any of the bytes it declared. Two bytes, `\xff\x7f`, commit 16,383 written bytes and the connection then waits out the 5-second substream-open timeout. Ten thousand concurrent connections carrying 20,000 attacker bytes in total raised the victim's resident set by 216,496 kB; against a silent control of 5,856 B per connection, 16,313 B per connection is attributable to this line, 99.6% of the 16,383 predicted from source. Total RSS rose 86%, from 250,904 kB to 467,400 kB, in a single 1.89-second burst, and did not fall when the connections expired, because the allocator retains the freed 16 KiB chunks. Pending connections are never registered against the established-connection limit, so the ceiling is the operator's file-descriptor limit — increasing network processing node resource consumption by far more than 30% without brute force actions.

IMM-BC-MED-88249

ExtrinsicBaseWeight still prices sr25519 after the switch to ML-DSA, undercharging every signed extrinsic

Report ID	#88249
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Inaccuracies to the weight (similar to gas) accounting system that under or overcharge by more than 50% of the correct amount (and are not intentional as indicated by the program's terms)

Description

base_extrinsic is the per-extrinsic charge covering decoding and signature verification. [configs/mod.rs#L86-L89](#) builds the block weights with **BlockWeights::with_sensible_defaults**, which routes through the builder and seeds every dispatch class from the inherited constant ([limits.rs#L350](#)). That constant is upstream's:

Its generated header records **RUNTIME: Development** on **substrate-node**, benchmarked on 2024-11-08 on an Intel Xeon at 2.60 GHz — a chain that signs with sr25519. [src/lib.rs#L113](#) replaces the signature type with **DilithiumSignatureScheme**, so every signed extrinsic now performs an ML-DSA verification inside WASM, and the project's own benchmarking harness signs ML-DSA-87.

The constant was never re-derived. Nothing under **runtime/** overrides it, and **scripts/regenerate_weights.sh** runs only **benchmark pallet**, never **benchmark overhead** — the subcommand that produces it, and which does not run as shipped. Running it against this runtime measures 871,694 ns per extrinsic on the fastest of three hosts against the 108,157 ns the runtime declares: the runtime charges 12.4% of the correct amount, an undercharge of 87.6%. The other two hosts give 91.6% and 94.2%, and a like-for-like **production** build accounts for only 6.4% of the gap. The threshold for this impact class is 50%. The shortfall is added to every extrinsic in every dispatch class, so block accounting and fees are wrong in the same proportion — inaccuracies to the weight accounting system that undercharge by more than 50% of the correct amount.

IMM-BC-MED-88218

Inherited sr25519 base weight leaves Dilithium verification unmetered in blocks and unpriced in the pool

Report ID	#88218
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Causing network processing nodes to process transactions from the mempool beyond set parameters

Description

`BlockWeights` is the ledger that bounds how long a block takes to execute. `CheckWeight` adds the per-class `base_extrinsic` to every extrinsic's weight, and that charge is the only budget covering signature verification — the separately benchmarked extension weights never touch the signature:

([check_weight.rs#L163-L166](#)). Quantus never overrides it, so it stays at upstream's sr25519 figure of 108,157 ns, while each extrinsic actually runs a Poseidon hash over the full public key and an ML-DSA verification in WASM with no host function ([traits.rs#L160-L184](#)).

That same verification also runs during pool validation, where `uxt.check(...)` executes before any nonce, solvency or fee check ([src/lib.rs#L983-L991](#)). `benchmark_overhead` measures 1,340,574 ns per extrinsic against the declared 108,157 ns, a factor of 12.39; even the minimum across 100 repetitions, 1,261,221 ns, is 11.7 times the declared value. A block of 541 extrinsics — the count the tool derived by building one — records 58.5 ms of weight while taking 725.3 ms to execute, leaving 666.7 ms per block that the block-weight limit cannot see. In the pool path nothing is charged at all: throwaway Dilithium keys signing transactions with a nonce that will fail impose the full verification on every node, causing network processing nodes to process transactions from the mempool beyond set parameters.

Severity note. The report states its own limit: this is not fee arbitrage. `LENGTH_FEE_MULTIPLIER` charges 1,000,000 planck per byte and `WeightToFee` is `IdentityFee`, so a 7,266-byte extrinsic pays roughly 7.27 ms-equivalent in length fees for about 1.34 ms of real base work — the signature envelope over-covers its own verification by about five times, and no attacker buys in-block compute below cost. The defect is confined to the weight ledger and to the fee-free pool path.

IMM-BC-MED-88397

Self-directed reversible transfers record wormhole ZK-tree leaves for credits that never occur

Report ID	#88397
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	A bug in the respective layer 0/1/2 network code that results in unintended smart contract behavior with no concrete funds at direct risk

Description

Every balance credit becomes a leaf of the wormhole ZK tree, created either by the post-dispatch event scanner or by an explicit recorder call. The scanner matches `Balances::TransferOnHold` unconditionally and turns it into a leaf ([transaction_extensions.rs#L360-L365](#)); its only exclusion anywhere is a vesting-pot address comparison inside the `Transfer` arm. Cancelling a one-time schedule takes the branch where `pending.guardian` is the sender, so `release_held_funds_with_fee` issues `transfer_on_hold` from the sender to the sender ([src/lib.rs#L943-L951](#)), a primitive with no `source == dest` short circuit that emits the event anyway. Separately, `do_execute_transfer` records a leaf on `post_info.is_ok()` alone ([#L721-L732](#)); a schedule pointed at the sender's own address makes the inner `transfer_keep_alive` take its self-transfer early return, move nothing, and still return `Ok`.

Neither mechanism checks that value moved. `record_transfer_proof`, the chokepoint both pass through, already drops one class of value-less credit for exactly this reason — a leaf for it "is pure state growth" — but filters on the amount and never on the accounts:

Anyone with a funded account therefore writes arbitrary credit into the wormhole deposit record for a transaction fee. Tests driven through `Executive::apply_extrinsic` record one leaf per cancel carrying the full scheduled amount while the sender's balance falls only by the fee and the named destination receives nothing; five rounds insert five such leaves. The leaves are not spendable, but the tree permanently asserts credits that never happened, a `NativeTransferred` event fires for each, and `TransferCount` advances — unintended behaviour in the layer-1 code with no concrete funds at direct risk.

IMM-BC-MED-90047

Unbounded SURI path depth in `derive_dev_account` bypasses the dev-account cap through the public light-client protocol

Report ID	#90047
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

Every full node registers an inbound light-client request handler ([builder.rs#L1185-L1191](#)), which forwards a peer-selected method name and bytes to `execution_proof` with no runtime-method allowlist.

[apis.rs#L323-L332](#) exports `GenesisBuilder::build_state` from the production runtime, and the genesis builder omits input-size limits on the stated premise that the API "is not reachable by network peers or on a running chain". `derive_dev_account` caps the account count at `MAX_DEV_ACCOUNTS = 1_000` because each derived account performs an expensive sr25519 derivation:

The cap bounds the count and nothing else. The function otherwise validates only a non-zero count, a balance above the existential deposit, and a template containing `{}`; the per-account loop then parses the whole SURI and folds one hard derivation per path junction ([src/lib.rs#L1368-L1394](#)), with no bound on template bytes or junction count. A 34,313-byte request carrying 1,000 accounts and a template of 10,922 junctions requests 10,922,000 derivations. Measured against an unmodified node over the real `/<genesis>/light/2` protocol, one such request occupied the serial handler for 202.5 seconds and consumed 189.95 CPU-seconds, 93.73% of one core against a 0.87% idle control. The queue is node-global and serial, so a normal request from a second peer — 10.060 ms at idle — timed out at 15,007 ms and was still executed 187,847 ms after submission; cancelling the stream, dropping the peer and the protocol timeout all leave the work running. One request every ~3.4 minutes, about 169 bytes per second, keeps the service permanently occupied: damage to the users and the protocol with no profit motive for the attacker.

IMM-BC-MED-88227

Degenerate-key filter in ML-DSA `verify_var` misses the `t1` family that also drops the challenge term

Report ID	#88227
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-gp-rusty-crystals/tree/audit-comp-ready/dilithium
Impact	A bug in the respective layer 0/1/2 network code that results in unintended smart contract behavior with no concrete funds at direct risk

Description

Verification recomputes $w1' = \text{UseHint}(h, Az - c \cdot 2^D \cdot t1)$, and the crate's own test documents the property at stake: a successful verify must imply possession of a real secret key. `verify_var` therefore rejects the degenerate all-zero `t1` public key, with a comment explaining that when the $c \cdot 2^D \cdot t1$ term vanishes an attacker can forge $(z = 0, \text{empty hint}, c = H(\mu \parallel w1\text{Encode}(0)))$ with no secret key:

([sign.rs#L707-L714](#)). The same predicate guards `PublicKey::from_bytes` ([frontend.rs#L425-L431](#)).

The attack is identified correctly and the precondition is not. `HighBits(x)` is zero for every `x` whose centred representative satisfies $|x \bmod \pm q| \leq \text{GAMMA2}$, so the challenge drops out of the relation for a whole family of non-zero `t1`, not only for zero. With `D = 13` and `GAMMA2 = (q-1)/32 = 261,888`, a single coefficient of value 1 to 31 qualifies, and because $2^{13} \cdot 1023 = q - 1 \equiv -1$, so does the top of the 10-bit range. Exhaustive enumeration over one non-zero coefficient accepts 63 of 1023 values on ML-DSA-87 and ML-DSA-65 and 23 on ML-DSA-44, matching the predicate exactly. The coefficient works in any row and any of the 256 positions and `rho` is unconstrained, so the forgeable family is of order 2^{256} . The forgery is one deterministic pass through the public API and verifies for every message and every context, while the same signature against an honestly generated key is rejected. The chain's only signature scheme accepts signatures produced without a private key — unintended behaviour with no concrete funds at direct risk, since reaching an existing account would require a Poseidon preimage.

IMM-BC-MED-88386

txWatch subscription tasks outlive unsubscribe, so RPC caps do not bound retained tasks

Report ID	#88386
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Increasing network processing node resource consumption by at least 30% without brute force actions, compared to the preceding 24 hours

Description

TxWatch is merged into every full node's RPC module and classified as Safe, on the stated basis that its work is bounded by a fixed broadcast capacity ([rpc.rs#L111-L116](#), [#L144](#)), so it stays available when a public node denies unsafe calls. Each accepted subscription takes a broadcast receiver and spawns a detached task that awaits only the channel:

([txwatch.rs#L151-L160](#)). The sink is inspected only after a transfer has already arrived.

Closing the JSON-RPC sink, on **txWatch_unwatchAddress** or on disconnect, does not wake **broadcast::Receiver::recv()**, and the task never awaits the cancellation-aware **closed()** future. The connection and subscription slots are released while the task, its receiver and its closed sink handle stay allocated, so the same client reconnects and allocates another full set: **--rpc-max-connections** and **--rpc-max-subscriptions-per-connection** bound live state, not cumulative retained tasks. With the server built at one connection and 256 subscriptions, 64 sequential cycles that explicitly unsubscribe everything and disconnect retain all 16,384 receivers and raise in-process RSS from 16,089,088 to 46,383,104 bytes, 188.3%, in 2.93 seconds. Against an unmodified **quantus-node --dev** capped at one connection, 64 cycles took 2.12 seconds and moved the process physical footprint from a settled 71 MB to 157 MB, 121.1%, still 163 MB ten idle seconds later — increasing network processing node resource consumption by well over 30% without brute force actions.

IMM-BC-MED-89154

Sparse public wormhole batches retain the full 742-exit weight declaration after settling one exit

Report ID	#89154
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Block stuffing

Description

The level-2 circuit deliberately permits a public batch holding one real private-batch proof and 52 reusable dummies, and the pallet skips the inert segments and settles the single real exit. `verify_public_batch` declares `WeightInfo::verify_public_batch()` ([src/lib.rs#L620-L626](#)), which scales the storage tail and the per-exit ZK-tree hashing by `public_batch_max_exits()` — the structural maximum of $2 \times 7 \times 53 = 742$ ([weights.rs#L184-L197](#)). As an upfront maximum that is safe.

The success path, however, never reduces it:

`actual_weight: None` tells FRAME to retain the entire declaration, so a dispatch that performed one exit and skipped 52 inert segments is charged for 742. A genuine 223,616-byte proof that passes pool validation at the fixed unsigned priority and the full ZK gate in `pre_dispatch` settles exactly one exit leaf and retains 3,744,227,000,000 ps of call weight — 83.205% of the 4.5-second Normal quota — while the call is unsigned and `Pays: No`. The same one-exit settlement through the private-batch path reserves 87,137,000,000 ps, so the supported public format is a 42.97× weight-reservation amplifier. One sparse proof per block leaves 755.773 ms, enough for at most eight 87.137-ms private proofs, cutting same-priority wormhole exit capacity from roughly 24 to 8 with no malformed proof, verifier bypass, colluding miner or brute-force traffic: block stuffing.

IMM-BC-MED-90343

Txpool skips maintenance on every finality event, so watched transactions are falsely timed out

Report ID	#90343
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

Pool maintenance on finality is the only path that retires pruned transactions and fires watcher `Finalized` statuses in the vendored pool that is every node's default. The enactment guard skips maintenance whenever the absolute height distance between an incoming chain event's block and `recent_best_block` exceeds `SKIP_MAINTENANCE_THRESHOLD = 20` ([enactment_state.rs#L107-L123](#)):

`force_update` records only `recent_finalized_block` on `Finalized` events ([#L192-L196](#)), so the comparison is always against the live best block. Quantus finalizes exactly `best_number - max_reorg_depth` after each import with `MaxReorgDepth = ConstU32<100>`, so every `Finalized` event arrives with a delta of about 100 and the guard — written to reject deep forward jumps during full sync — discards 100% of finality events in normal operation.

A watched transaction therefore stalls at `inBlock` forever. `MAX_FINALITY_WATCHERS = 512` then bounds the watch list, and each eviction fires `finality_timeout` carrying the evicted block hash ([listener.rs#L255-L263](#)) — the terminal signal integrators read as "dropped, will not finalize". On an unmodified node a transfer included at height 4 received `finalityTimeout` at best height 516 carrying its own inclusion block hash, while the node's finalized height was 416; `author_unwatchExtrinsic` then returned false, so the false terminal status is the last word the integrator receives. No attacker action is required, and timeout-driven resubmission executes the same payment twice: damage to the users and the protocol with no profit motive for an attacker.

IMM-BC-MED-88736

Pre-authentication ML-KEM/ML-DSA Noise work is uncapped, letting unauthenticated peers inflate node CPU and memory

Report ID	#88736
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Increasing network processing node resource consumption by at least 30% without brute force actions, compared to the preceding 24 hours

Description

Quantus vendors litep2p and replaces its Noise handshake with a post-quantum pqXX variant keyed by an ML-KEM-768 static key. On every inbound TCP connection, ``negotiate_connection`` completes the `/noise` multistream-select negotiation and then calls `noise::handshake`, which builds a `NoiseContext`: ``NoiseContext::new`` generates a fresh ML-KEM-768 keypair and builds a Clatter session, and assembly signs the KEM public key with the node's ML-DSA identity key — all before the peer's first pqXX message is read. The reader then takes a 2-byte length and allocates that many zeroed bytes before the payload arrives:

This pre-authentication work is uncapped. ``ConnectionLimits::on_incoming`` counts only the established `incoming_connections` set, so pending handshakes never count against `max_incoming_connections`, and the length prefix is honoured up to `0xffff`. A remote unauthenticated peer that sends only the 28-byte `/noise` negotiation therefore forces one ML-KEM-768 keygen and one ML-DSA signature per connection, and two extra bytes force a 65,535-byte allocation held until the handshake timeout. On the release node, 512 stalled connections raised process CPU from 0.11% to 13.5% and 4,096 connections added roughly 500 MB RSS against a few KB/s of attacker traffic, increasing network processing node resource consumption by far more than 30% without brute force actions.

IMM-BC-MED-90210

Default transaction-pool size exceeds the gossip known-hash cache, causing repeated full-peer disconnects and resource amplification

Report ID	#90210
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Increasing network processing node resource consumption by at least 30% without brute force actions, compared to the preceding 24 hours

Description

To accommodate its ~7.3 KB post-quantum transactions, Quantus raises the compiled transaction-pool default to 256 MiB, selected automatically when a node starts without `--pool-limit`. The transaction-gossip protocol independently retains only a fixed 10,240-entry per-peer known-hash cache, and that cache is an `LruHashSet` whose limit is exclusive:

The pool default now sits above that boundary. Once the ready pool holds 10,240 transactions, the cache evicts the second hash as the first is inserted, so every hash misses on every 2.9-second propagation scan and the node resends the whole pool to each full peer. That floods the notification queue, and `force_close` then closes the entire transport connection — including block announcements — even when other protocols have open substreams. The peer reconnects with an empty cache and the cycle repeats. Using ordinary signed, fee-paying transactions submitted once, the reporter measured a 159% victim CPU increase and roughly doubled RSS with block and transaction streams closing repeatedly, increasing network processing node resource consumption well beyond 30% without brute force actions.

IMM-BC-MED-88329

Missing guardian consent in `set_high_security` lets one signer exhaust any guardian's bounded quota

Report ID	#88329
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

In `pallet-reversible-transfers`, each guardian's wards live in `'GuardianIndex'`, a target-keyed `BoundedVec` capped at `'MaxGuardianAccounts = 32'`. The permissionless `'set_high_security'` nominates any account as guardian and appends the caller into that guardian's index:

The call authenticates only the nominator; it checks the target is not the caller but never obtains authorization from the guardian before consuming its capacity. Because `'Utility::as_derivative'` deterministically rewrites the signed origin to a keyless, zero-balance pseudonym that need not exist on chain, one ordinary account can batch 32 `as_derivative` calls, each targeting the same guardian, and fill all 32 slots in a single signed transaction. After the batch, any legitimate account that selects that guardian is rejected with `TooManyGuardianAccounts`. The guardian cannot repair the quota: the pallet has no eviction path, and `recover_funds` deliberately preserves both `HighSecurityAccounts` and `GuardianIndex`. The reporter measured the full attack at 0.023 QNT, permanently denying any chosen address the guardian feature — griefing with no profit motive but lasting damage to users and the protocol.

IMM-BC-MED-89524

`retarget_schedule` skips settlement inside the claim rate-limit window, reassigning the outgoing beneficiary's vested funds

Report ID	#89524
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

`pallet-vesting`'s `retarget_schedule` is documented to "Settle any payout a permissionless claim could currently force, then change the beneficiary. This makes retargeting independent of claim transaction ordering." It settles through `claim_plan`, then swaps the beneficiary:

However, `claim_plan` computes the full payable amount and only then returns `TooSoon` when `last_claim_at` is within `MinClaimInterval`. `TooSoon` is a rate limit, not an absence of funds, yet `retarget_schedule` collapses it to zero and never advances `schedule.claimed`. On the live runtime `MinClaimInterval` is `MILLIS_PER_DAY`, and `claim` is permissionless, so any unrelated account can stamp `last_claim_at` on any schedule and hold it inside that window indefinitely for one transaction per schedule per day. Whenever the treasury then performs a routine rotation, the outgoing beneficiary is paid nothing and their accrued entitlement — up to one day of linear vesting — transfers to the incoming beneficiary at the next successful claim. The emitted event carries `vested_paid: 0`, indistinguishable from a genuine nothing-to-claim, and the loss multiplies across a grantee's schedules: a silent, permissionless griefing of beneficiary funds with no profit to the actor who arms it.

IMM-BC-MED-90408

Safe RPC `state_call` exposes `GenesisBuilder` with an unbounded derivation template, pinning node CPU cores

Report ID	#90408
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Increasing network processing node resource consumption by at least 30% without brute force actions, compared to the preceding 24 hours

Description

Quantus's production runtime exports the full `'GenesisBuilder::build_state'` API, which an unauthenticated client can invoke through the standard blocking `state_call` RPC even under `--rpc-methods safe`. The caller supplies `devAccounts = (count, balance, derivation_template)`. The [account-count cap](#) limits `num_accounts` to 1,000, but the [derivation loop](#) validates only that the template contains `{}`:

However, the template's length and path-junction count are unbounded, and every accepted account reparses the whole path and performs the requested sr25519 derivations, so runtime work grows as roughly `account count × junction count`. A template of `"//{"` repeated 1,000 times fits in an 8,739-byte request yet occupied one core for about 6.7 seconds; four concurrent clients raised the frozen node from ~100% to ~500% process CPU at only ~5.2 KB/s, and 10,000 junctions produced minute-scale jobs from ~80 KB. Closing the client does not cancel the admitted blocking work. The attack needs no account, signature, fee, or high request rate, increasing network processing node resource consumption by far more than 30% without brute force actions.

IMM-BC-MED-88215

Any account can permanently consume another address's bounded guardian index with no consent or removal path

Report ID	#88215
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

In `pallet-reversible-transfers::set_high_security` writes the caller into the target guardian's `'GuardianIndex'`, a `BoundedVec` capped at `'MaxGuardianAccounts = 32'`:

The `write_site::try_pushes` the caller without any consent, signature, deposit, or even on-chain existence of the account named as guardian. `GuardianIndex` has exactly two write sites — this one and the genesis builder — and both are push-only: no extrinsic, hook, or helper ever removes an entry, and enrollment is [documented as a one-way operation that cannot be undone](#). These properties compose: any unprivileged party can spend 32 transactions to permanently exhaust the guardian index of any chosen address, after which every honest user who names that address receives `TooManyGuardianAccounts` and is left with no high-security protection. Because the guardian need not exist on chain, a published address — a custody provider, an institutional cold-wallet guardian, a well-known community address — can be pre-squatted before its owner ever transacts. The victim has no in-protocol remedy: even `recover_funds` leaves the index untouched, so restoring capacity requires a runtime upgrade plus a storage migration. This is griefing — no attacker profit, permanent damage to users and to the pallet's core protective feature.

IMM-BC-MED-88671

Permissionless vesting `claim` lets a third party fragment a grant into rounding-lossy payout leaves

Report ID	#88671
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

`pallet-vesting`'s `claim` is permissionless: it discards the `ensure_signed` result and lets any signer trigger a payout on any schedule, paying the largest amount vested at call time via `claim_plan`:

Each payout is settled through `pay_out`, which records a separate Wormhole transfer-proof leaf, and each leaf loses value independently to the withdrawal circuit's per-leaf fee rounding. Because the claim is permissionless, the choice of fragmentation belongs to an attacker, not the beneficiary: the one-day `MinClaimInterval` bounds only the rate, so an unrelated account can call `claim` once per interval and split a grant into many small leaves, multiplying the rounding loss rather than amortising it. Aggregation cannot recover it, since the inequality is enforced per leaf before batch aggregation. For a 365-day, 365-QUAN linear grant, daily third-party claims produce 365 one-QUAN leaves whose combined maximum output falls from 364.85 to 361.35 QUAN — about 3.50 QUAN permanently destroyed for the beneficiary, scaling with schedule duration. The attacker gains nothing and needs only transaction fees: griefing.

IMM-BC-MED-89213

Missing account-reaping cleanup leaves dead entries permanently occupying a guardian's bounded index

Report ID	#89213
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

In `pallet-reversible-transfers`, the permissionless `set_high_security` is the only extrinsic that writes `'GuardianIndex'`, a `BoundedVec` capped at 32, and it is append-only — no `retain`, `remove`, or `clear` exists:

When `frame_system` reaps an account it fires `'on_killed_account'`. However, the runtime's `'frame_system::Config'` does not override `OnKilledAccount`, so no reversible-transfers cleanup runs on reaping, and `recover_funds` intentionally [preserves the high-security status and guardian relationship](#). An attacker funds 31 throwaway accounts, enrolls each against a victim guardian to fill the index to 32/32, then lets the guardian drain and reap them via `recover_funds`. The accounts die, but their `GuardianIndex` and `HighSecurityAccounts` entries persist as orphans occupying the slots forever. Any honest user who then names that guardian receives `TooManyGuardianAccounts` and gets no protection, permanently. The only remediation is a runtime upgrade adding the hook plus a storage migration. The attacker gains nothing and can reduce cost to a few UNIT plus fees: griefing that permanently degrades the guardian feature for the targeted address.

IMM-BC-MED-89889

Permissionless vesting `claim` strands a schedule below the termination threshold, blocking the treasury's clawback

Report ID	#89889
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

`pallet-vesting`'s `claim` accepts any signer for any schedule. For a non-final payout, `claim_plan` caps the amount at `remaining - MinimumPayout`, guaranteeing the schedule keeps one minimum-sized *total* remainder — but not that the *already-vested and unpaid* portion is either zero or at least `MinimumPayout`. However, `end_schedule` rejects exactly that middle case:

When the uncapped candidate exceeds the cap by less than one minimum payout, the permissionless claim succeeds and leaves a non-zero sub-`MinimumPayout` vested remainder, so a schedule that was terminable one instant earlier can no longer be ended. Any unrelated funded account can call `claim` at that point just before the administrator's termination, permanently denying the treasury its clawback of the final unvested tranche — bounded below one `MinimumPayout` (0.99 QUAN at production values) regardless of schedule size — and blocking termination until natural maturity, by which point the amount vests to the original beneficiary instead of returning to the treasury. For a 100-QUAN one-year grant the block lasts about 3.6 days. The attacker needs one signed extrinsic and gains nothing: griefing of the treasury's clawback path.

IMM-BC-MED-89307

Sub-existential-deposit block fees are minted before the account-endowing reward and diverted to the treasury

Report ID	#89307
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

`pallet-mining-rewards` states that 100% of transaction fees go to the block miner. In `'on_finalize'` it mints the collected fees to the miner *before* the base reward:

`'mint_reward'` credits the exact amount with `mint_into`, which fails with `BelowMinimum` when the credit would leave a not-yet-existing account below the existential deposit. However, on the block a fresh miner first wins, its author address has no prior balance, so a sub-existential-deposit fee total fails on this first credit and the error arm redirects the fee to the treasury and records a wormhole proof crediting the treasury. The base reward on the very next line then creates the account, but the fee is already gone; `CollectedFees` was cleared at the top of the hook and no retry path returns it. An attacker forces this on demand: a post-dispatch `Pays::No` extrinsic collapses the charged fee to a sub-ED tip, so targeting each new miner's first block denies it fee revenue at a cost of only the tip. No attacker profit, permanent loss to the miner: griefing.

IMM-BC-LOW-88568

Transaction payment weight omits the custom fee collector's storage access, undercharging every paid extrinsic

Report ID	#88568
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Inaccuracies to the weight (similar to gas) accounting system that under or overcharge by less than 50% of the correct amount (and are not intentional as indicated by the program's terms

Description

The runtime binds transaction payment to a custom fee handler and, in the same configuration block, selects a generic upstream benchmark for that extension at [configs/mod.rs#L477-L490](#). That benchmark, [weights.rs#L81-L90](#), declares storage `System::Account (r:1 w:1)` only. Every paid signed extrinsic, however, also runs `TransactionFeesCollector`, which mutates `CollectedFees` and reads it back for the `FeesCollected` event at [src/lib.rs#L210-L218](#); that additional unique key appears nowhere in the declared weight. Under `RocksDbWeight` ([configs/mod.rs#L124](#)) the omission is a fixed 125,000,000 ps. Against the 622,787,000 ps the runtime declares for a minimal signed `System::remark`, measured by the report's production-runtime test, the payer and the block meter are undercharged by 16.7160% — an inaccuracy in the weight accounting system of less than 50% of the correct amount.

IMM-BC-LOW-89098

`count_transfers` has no `Multisig::execute` arm, so wrapped transfers carry zero proof-recording weight

Report ID	#89098
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Inaccuracies to the weight (similar to gas) accounting system that under or overcharge by more than 50% of the correct amount (and are not intentional as indicated by the program's terms)

Description

`WormholeProofRecorderExtension` prices proof recording statically, per transfer, and derives the count by walking the submitted call in `count_transfers` ([transaction_extensions.rs#L217-L232](#)). `Multisig::execute` carries only `(multisig_address, proposal_id)` and reads its payload from storage, so the walker cannot see the inner transfers: the call falls through to `_ => 0` at [#L293](#) and `weight()` returns `Weight::zero()` ([#L414-L421](#)). The dispatch records a leaf for every inner transfer anyway, and the shortfall reaches the block only afterwards through `register_extra_weight_unchecked` ([#L508-L517](#)), on a path the runtime itself marks "not fee-charged" ([#L475-L477](#)). The fee never recovers it. By the report's arithmetic over the cited constants, an ordinary one-transfer multisig payment collects 0.46 ms against a correct 5.34 ms, 91% of the charge missing, and the same transfers submitted as a `Utility::batch` pay in full.

Severity note. The behaviour is deliberate and documented in the source: the comment above the match names `Multisig::execute` as a wrapper whose inner call cannot be counted statically and states that its proof-recording work is reconciled in `post_dispatch`. Block-capacity accounting therefore stays sound, and the weight-inaccuracy impact is qualified on the shortfall not being intentional.

IMM-BC-LOW-90448

`recover_funds` weight counts pending transfers but not the scheduler-agenda density its cancellations pay for

Report ID	#90448
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Inaccuracies to the weight (similar to gas) accounting system that under or overcharge by less than 50% of the correct amount (and are not intentional as indicated by the program's terms)

Description

`ReversibleTransfers::recover_funds` derives its post-dispatch weight from a single runtime parameter, `num_processed`, the number of the sender's pending transfers ([src/lib.rs#L553](#), returned at [#L633](#)). The expensive part of each iteration, `T::Scheduler::cancel_named` ([#L583](#)), instead scales with how many tasks share the target agenda key; the benchmark advances a block between every schedule to keep agendas at one entry ([benchmarking.rs#L231-L236](#)), so the constants at [weights.rs#L326-L335](#) carry a per-transfer term and no per-agenda term. Scheduling in one block clusters every transfer under one key, and the PoC reports byte-identical `actual_weight` for eight keys and for one. Its micro-measurement fits 807 ns per extra agenda entry, which at the `LOWEST_PRIORITY` cap of 40 entries undercharges by 5.94% of the amount charged per transfer.

IMM-BC-LOW-90496

Unreserved tips let one account fill a node's future queue and evict every lower-priority transaction

Report ID	#90496
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

`ChargeTransactionPayment::validate` computes a transaction's pool priority from its tip after checking only that the signer could withdraw `fee + tip` once; the withdrawal itself happens later, in `prepare` ([src/lib.rs#L1019-L1045](#)). A nonce-gap transaction never reaches `prepare`, so one unchanged balance backs an advertised tip on every entry a signer places in the future queue, whose default capacity is one tenth of the 36,772-transaction pool limit ([builder.rs#L71-L81](#)). Quantus evicts the lowest-priority future entry first ([base_pool.rs#L530-L565](#)). The runtime PoC signs all 3,677 transactions needed to fill that queue at a 500-QUAN tip each from one 1,000-QUAN account whose nonce and balance are unchanged afterwards; the pool test then shows a lower-priority victim returned `ImmediatelyDropped` and banned ([validated_pool.rs#L296-L329](#)), for the default 30 minutes.

Severity note. Several boundaries stated in the report limit the denial: ready transactions are unaffected, each node must be reached directly, an operator can clear the pool, and the control test in the same PoC shows that a later transaction matching the attacker's priority displaces it. The denial therefore reaches only strictly lower-priority future entries on nodes the attacker submits to.

IMM-BC-LOW-88830

Private-batch circuit does not bind a real positive-value output to a non-zero exit account

Report ID	#88830
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-gp-zk-circuits
Impact	Modification of transaction fees outside of design parameters

Description

The project's own specification records that `exit_account_1/2` are unconstrained at the leaf and that the binding obligation lives in the private-batch relation ([SPEC.md#L180-L184](#)), and the leaf gadget itself adds no constraints ([substrate_account.rs#L144-L150](#)). At the private-batch layer, the gadget masks a dummy child's exit and amount to zero and then groups slots by exit account, but for a real child `is_dummy_i` is false and `exit_raw` is forwarded unchanged, with nothing requiring a non-zero exit when the amount is non-zero ([circuit_logic.rs#L329-L377](#)). The researcher's executed test proves a real leaf with `output_amount_1 = 1` and a zero exit account, aggregates it, and verifies the aggregated proof, which exposes `slot[0]: amount=1, exit_account=[0,...,0]`. A positive-value output therefore reaches the settlement layer inside a valid proof, bound to the canonical destination the circuit otherwise reserves for unused slots.

IMM-BC-LOW-89077

Genesis wormhole leaves record `total_balance`, so reserved funds are represented as exitable value

Report ID	#89077
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	Inflation bugs (unauthorized minting)

Description

At block one, `pallet-wormhole`'s `on_initialize` walks every account and records a ZK leaf for each non-zero balance, so that the wormhole tree opens with a leaf representing each account's value. The amount it records is `Currency::total_balance` ([src/lib.rs#L478-L488](#)), which includes reserved and locked funds, while an exit is credited with `Unbalanced::increase_balance` ([#L828-L832](#)) as free balance, without reducing or unlocking the reserve on the source account. The leaf therefore represents more value than the account can spend. Where such a leaf is spendable, the reserve stays held on the original account while an equivalent amount of free balance is minted to the exit account, and every economic lock counted in reserved balance is bypassed for the value that left through the wormhole.

Severity note. The researcher's own follow-up narrows the exploitability: a leaf can only be spent by the holder of the secret behind a wormhole deposit address, so leaves recorded for ordinary genesis accounts are permanently un-nullifiable and no minting path exists on the chain as configured. What is confirmed today is the un-nullifiable-leaf state bloat and a latent flaw that a genesis preset allocating reserved balances to secret-controlled deposit addresses would activate. Neither unauthorized minting nor direct loss of funds is reached on the current configuration.

IMM-BC-LOW-89310

Asymmetric difficulty retarget lets an author-supplied timestamp book a deficit later blocks cannot repay

Report ID	#89310
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	A bug in the respective layer 0/1/2 network code that results in unintended smart contract behavior with no concrete funds at direct risk

Description

`QPow::calculate_difficulty` retargets from a single block-to-block timestamp delta, documenting zones in which a fast block raises difficulty by 1/2048 and a block between one and two divisors leaves it unchanged. The move is asymmetric: `max(1 - time_factor, -99)` reaches -99 downward but is capped at +1 upward ([src/lib.rs#L230-L232](#)), the delta is the author's own timestamp minus the previous author's, the inherent accepts 30,000 ms of future drift ([src/lib.rs#L306-L321](#)), and that inflated timestamp becomes the next baseline ([src/lib.rs#L176](#)). One drifted block takes -4 while the squeezed blocks that follow return +1 each. The PoC folds 4,000,000 down to 3,998,038 over 48,000 ms of wall clock that four honest 12,000 ms blocks leave unchanged, and 100 cycles reach 3,808,556. The pallet has no cumulative-elapsed-time or median-timestamp check, so the deficit compounds, raising the block rate and pulling forward the per-block emission and every block-counted governance and vesting deadline.

Severity note. The magnitude the report establishes is small and slow: one cycle books a 1,962 deficit against a 4,000,000 difficulty, and the decay accrues only for as long as the attacker keeps winning and authoring blocks. The retarget also floors the author-supplied delta at `MIN_RETARGET_BLOCK_TIME_MS` so an implausibly small timestamp cannot steer it further.

IMM-BC-INSIGHT-88243

`check_wormhole_path` validates only the coin type and the crate exports no path constructor

Report ID	#88243
Severity	Insight
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-gp-rusty-crystals/tree/audit-comp-ready/hdwallet
Impact	—

Description

`check_wormhole_path` accepts any depth and any arrangement of a derivation path's segments, requiring only that segment 2 is the wormhole chain ID ([src/lib.rs#L421-L427](#)), and the crate exports the coin-type constants with no path constructor beside them ([#L85-L88](#)). Each consumer therefore invents its own layout: the node CLI varies the account segment and the mobile SDK varies the address-index segment, so the two derive unrelated wormhole keys for the same mnemonic at every index except 0.

IMM-BC-INSIGHT-88290

PeerId is derived from the sender's raw protobuf bytes rather than from the decoded public key

Report ID	#88290
Severity	Insight
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-quantus-chain
Impact	—

Description

`parse_and_verify_peer_id` hands one `identity` slice to two consumers with different semantics: `RemotePublicKey::from_protobuf_encoding` decodes it through `prost` and verifies the signature against the decoded key, while `PeerId::from_public_key_protobuf` hashes the literal bytes received on the wire ([peer_id.rs#L68-L77](#)). Because protobuf admits several wire encodings of one message, a single Dilithium key yields several valid `PeerIds`, and the `Peerstore` ban table keyed on `PeerId` ([peerstore.rs#L180-L182](#)) reads a re-encoded identity as a peer it has never seen.

Findings – Websites & Apps

IMM-WA-CRITICAL-89007

Lock screen leaves the underlying wallet in the accessibility tree, letting a bound service sweep the wormhole balance

Report ID	#89007
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Direct theft of user funds

Description

The mobile wallet gates use of the application behind a local-auth lock. `AuthWrapper` watches the local-auth state and, whenever the wallet is not authenticated, paints its lock screen in place of the wallet ([auth_wrapper.dart#L16-L21](#)). The same design treats a fresh biometric as the precondition for acting on funds: seven call sites across the send, multisig and settings flows call `LocalAuthService().authenticate()` before proceeding, and the lifecycle manager records that "the send, multisig and settings flows authenticate through `LocalAuthService` directly" ([app_lifecycle_manager.dart#L111-L118](#)). While the lock screen is displayed, the wallet beneath it is meant to be inoperable.

`app.dart` mounts the lock as a plain `Stack` sibling painted over the routed child:

```
builder: (context, child) {
  return Stack(children: [child!, const AuthWrapper()]);
},
```

[app.dart#L48-L50](#)

The locked branch returns an ordinary opaque `Scaffold` ([auth_wrapper.dart#L40-L42](#)). It is not a pushed route, so it never receives the `ModalBarrier` that `ModalRoute` supplies, and `ModalBarrier` is the widget that wraps content in `BlockSemantics`. A search across `mobile-app/lib` and `cold-wallet-app/lib` for `BlockSemantics`, `ExcludeSemantics`, `AbsorbPointer` and `ModalBarrier` returns nothing; the tree's single `IgnorePointer` sits in [address_input_field.dart#L46](#) and is unrelated.

An opaque sibling hides the screen underneath and absorbs pointer events, but it does not remove that screen from the semantics tree, and accessibility actions reach a node's handler without passing through pointer hit-testing. Every actionable control on the locked-out screen therefore stays reachable to a bound

Android `AccessibilityService`. One control left reachable is the mining-rewards redeem flow, and it is the fund-moving path with no re-authentication on it: `_redeem()` ([redeem_address_screen.dart#L82-L101](#)) pushes `RedeemProgressScreen`, whose `initState` calls `_startClaim()` ([redeem_progress_screen.dart#L37-L41](#)), which derives the wormhole key pair and calls `claimRewards` ([#L43-L66](#)). None of the seven `authenticate()` call sites lies on that path, while its closest sibling, [encrypted_send_strategy.dart#L137](#), spends the same secret from the same pool and does demand a fresh biometric.

`claimRewards` carries no amount parameter ([wormhole_send_service.dart#L145-L152](#)). `_runClaimFlow` enumerates every unspent transfer at the wormhole address and assigns each leaf's full net amount to the single caller-supplied destination ([#L250-L281](#)), bounded only by the `isValidSS58Address` check on the input field ([redeem_address_screen.dart#L67](#)). The pool is wider than the "mining rewards" label suggests: `deriveWormholeKeyPair` defaults to `index: 0` ([hd_wallet_service.dart#L72-L74](#)) and `EncryptedAccountService._deriveKeyPair` calls the same function on the same branch ([encrypted_account_service.dart#L160-L162](#)), so the rewards address is the encrypted account's external index 0, the first address the Receive screen shows. Malware holding only the Accessibility Service permission drives the locked wallet, writes its own destination into the recipient field, and exits the whole balance on chain in one irreversible step. This is direct theft of user funds.

Proof of Concept

1. A minimal Android `AccessibilityService` declaring `canRetrieveWindowContent` is installed and bound on the device; no root, no injected touch events and no physical access to the wallet are needed.
2. The wallet is cold-started and settles into its locked state, with `AuthWrapper` painting "Authorization required" over the wallet screen.
3. With the service disabled, a pointer tap at the covered control's coordinates fires no handler: the overlay absorbs touch, so the lock holds against an ordinary user.
4. With the service bound, the accessibility tree published while the lock screen is up still lists the underlying wallet's controls as clickable and enabled, including controls carrying no semantics label, which are reachable by index.
5. `ACTION_SET_TEXT` succeeds against the destination field while the app reports itself locked, writing an attacker-chosen SS58 address into the redeem recipient input.
6. `ACTION_CLICK` on the redeem control returns true and the application's own fund-moving handler executes while `lockVisible` is still asserted; the state persists for more than ten minutes with no timing window to win and no user interaction.
7. The identical widget tree with `BlockSemantics` wrapping the overlay removes the node from the semantics tree, so the harness separates a vulnerable build from a fixed one.

IMM-WA-CRITICAL-88730

Generated `MultiAddress::Index` codec consumes a compact integer the runtime does not, hiding `transfer_all` in a multisig proposal

Report ID	#88730
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/quantus_sdk
Impact	Direct theft of user funds

Description

The multisig approval flow is built to bind a co-signer's approval to the exact call the chain will execute. `MultisigService._withOnChainCallData` replaces indexer-supplied call data with the authoritative bytes from `Multisig.Proposals` storage, "so browsing surfaces (proposal rows, detail sheet) show what the chain will execute" ([multisig_service.dart#L293-L326](#)). The confirmation sheet decodes those bytes with `CallDecoder.decodeBytes` ([multisig_action_confirm_sheet.dart#L183-L195](#)) and renders the tree through `DecodedCallView` ([#L353-L389](#)), and the approve path resubmits those same bytes for the chain's byte-for-byte comparison ([multisig_approve_confirm_sheet.dart#L38-L53](#)). The property this is meant to deliver is that the call tree a signer reviews is the call tree the runtime dispatches for those bytes.

The runtime instantiates its address type with a unit index, `pub type Address = MultiAddress<AccountId, ()>` ([src/lib.rs#L135](#)), and looks accounts up through `AccountIdLookup<Self::AccountId, ()>` ([configs/mod.rs#L113](#)). For that instantiation `MultiAddress::Index()` encodes as its variant byte alone and consumes no value bytes. The generated Dart codec defines the same variant over `BigInt` and decodes it as a compact integer:

```
class Index extends MultiAddress {
  const Index(this.value0);

  factory Index._decode(_i1.Input input) {
    return Index(_i1.CompactBigIntCodec.codec.decode(input));
  }

  /// AccountIndex
  final BigInt value0;
```

[multi_address.dart#L152-L160](#)

The two schemas therefore disagree on how many bytes an `Index` address occupies, and nested `SCALE RuntimeCalls` carry no length prefix, so that disagreement moves the call boundary instead of producing a

decode error. One byte string is a complete, canonical call under both schemas with different structure and different semantics:

Offset	Bytes	Runtime reads	Application reads
0–1	09 06	Utility.if_else	Utility.if_else
2–4	0d 00 01	TechCollective.add_member(Index(()))	same call, Index variant
5–6	09 04	fallback Utility.force_batch	Compact(258) as the Index value
7–8	14 02	Vec length 5, first child pallet Balances	Wormhole.verify_private_batch
9–267	05 04 + 257	force_unreserve, then transfer_all and three remarks	compact length 257 and one opaque proof vector

Both decoders consume exactly 268 bytes, so trailing-byte and exact-fit checks do not detect the payload. A signer below the threshold proposes it; the honest co-signer's client fetches the authoritative bytes and renders `if_else(add_member(Account index 258), Wormhole.verify_private_batch(257 opaque bytes))`, with no `transfer_all`, no `force_batch`, no destination and no address-typed field. The chain's byte-equality check ([src/lib.rs#L779-L782](#)) passes because nothing was altered. At execution the runtime decodes the stored bytes with its own schema and dispatches them as the multisig account ([#L1123-L1156](#)): the privileged `add_member` main branch fails under the signed multisig origin, so `if_else` dispatches the fallback ([src/lib.rs#L568-L636](#)), and `force_batch` records the deliberately failing first child and continues with the rest ([#L451-L509](#)), executing `Balances.transfer_all(Id(attacker), false)`. Binding approval to authoritative bytes does not bind approval to the semantics shown to the signer, and the remaining honest signature is defeated. The PoC drains `1000 * UNIT`, the 2-of-2 multisig's full transferable balance, to the attacker key. This is direct theft of user funds.

Proof of Concept

1. A 2-of-2 multisig is created from real ML-DSA-65 keys and funded with `1000 * UNIT`; the attacker holds one of the two signer keys and cannot move the funds alone.
2. The attacker builds the 268-byte call whose `MultiAddress::Index` field lands on the schema disagreement, naming an attacker-controlled account as the hidden `transfer_all` destination, and submits it through `multisig.propose`.
3. The bytes read back from `Multisig.Proposals` are asserted byte-identical to the submitted payload.
4. The production Dart decoder used by the mobile proposal model is fed those exact bytes and yields `[if_else, add_member, verify_private_batch]`, with no `transfer_all`, no `force_batch`, no `Destination` label and no address-typed value.

5. The honest signer approves with those same bytes; the chain's byte-equality check passes and the proposal reaches its 2-of-2 threshold.
6. The attacker calls `multisig.execute`, and the runtime decodes its own `force_batch/transfer_all` tree and dispatches it as the multisig account.
7. The multisig's free balance is zero and the attacker's balance is `1000 * UNIT`; the same bytes run through the full signed-extrinsic pipeline, including signature validation and every transaction extension, produce the same transfer.

IMM-WA-CRITICAL-89999

Multisig discovery stores the indexer-supplied address without recomputing the derivation, letting a hostile indexer substitute the account

Report ID	#89999
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/quantus_sdk
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

A Quantus multisig's on-chain address is fully determined by its signers, threshold and nonce, and the SDK ships that derivation. `predict_multisig_address` computes the address as `hash(pallet_id || sorted_signers || threshold || nonce)` ([multisig.rs#L27-L48](#)), and a golden-vector test pins signers `5Grwva...` and `5FHneW...`, threshold 2, nonce 0 to `qzkvQ2YBa7RrvYKNCXxJw7hJZPBSi6GzZxXGN2fN6DddaH5HJ` ([#L75-L91](#)). Dart reaches it through `MultisigService.predictMultisigAddress` ([multisig_service.dart#L73-L81](#)). On the create path the wallet never trusts an externally supplied address: `resolveMultisigCreationParams` computes it itself for each candidate nonce ([#L87-L113](#), at [#L105](#)).

Discovery takes the other route. `discoverForUser` posts the discovery query and reads each record's address straight out of `record['id']` ([#L44-L68](#), at [#L57](#)), then hands the record to the mapper that builds the local account:

```
static MultisigAccount multisigAccountFromIndexerRecord(
    Map<String, dynamic> record, {
        required String myMemberAccountId,
        required String name,
    }) {
    final address = stringFromJson(record['id']);
    final creator = nestedAccountId(record['creator']);
    final signers = nonEmptyStringListFromJson(record['signers'], 'signers');
    final threshold = multisigThresholdFromJson(record['threshold'], signerCount: signers.length);
```

...

[multisig_service.dart#L163-L183](#)

The mapper reads the signers, the threshold and the nonce out of the same record — every input `predictMultisigAddress` requires — and stores `record['id']` verbatim as the account's identity. It never

calls the derivation, and no other point on the discovery path compares the reported address against it. The wallet authenticates the address it creates and accepts on trust the address it discovers, using the same data and the same primitive it already holds.

The accepted address then becomes the multisig throughout the application. `addMultisig` persists the discovered account ([discover_multisig_screen.dart#L26-L46](#)) through the accounts provider's `add` ([#L35](#)) and `SettingsService.addMultisigAccount`, which keys storage on `accountId` ([settings_service.dart#L269-L276](#)); the screen displays it to the user as the multisig's address ([#L217](#)); and the approve, execute and cancel confirmation sheets all key on `msig.accountId` ([multisig_approve_confirm_sheet.dart#L29](#), [multisig_execute_confirm_sheet.dart#L23](#), [multisig_cancel_confirm_sheet.dart#L25](#)). An attacker controlling the discovery indexer or its responses returns a record that lists the victim among the signers, so it reads as the user's own multisig, while the address field names an ordinary account the attacker holds. The signer list and threshold rendered in the interface are attacker-supplied display values; the address is the only field that decides where value settles, and the multiple-approval protection the interface implies does not constrain that account, because it is not a multisig. Deposits the user makes to their shared account, and funds a counterparty sends after the user shares the address, go to the attacker, and nothing in the wallet surfaces the discrepancy. The endpoint is served over HTTPS, so the precondition is indexer compromise rather than passive network interception. This is a malicious interaction with an already-connected wallet in which an attacker-controlled address is substituted for the user's multisig.

Proof of Concept

1. A loopback HTTP server stands in for the discovery indexer and answers the wallet's discovery query with one record whose signers include the victim, whose threshold is 2 and whose nonce is set, and whose address field holds an attacker-selected valid address that those inputs do not derive to.
2. The wallet's discovery endpoint is pointed at the loopback server; all network handling, parsing and account construction remain the wallet's own code, and only the server response is controlled.
3. `discoverForUser` is called for the victim, and the test asserts the wallet posted its real discover query by checking the request method and the query body.
4. Discovery returns the record successfully, without rejecting it.
5. The returned account's address is `qzqx7Uk9bJu5Eayk8LnG99Vt2DukQukoso7L99vs2yPnT5Ny5`, the attacker-selected value, and not the derived address for the same signers, threshold and nonce.
6. The derived control address `qzkvQ2YBa7RrvYKNCXxJw7hJZPBSi6GzZxXGN2fN6DddaH5HJ` is established independently by the wallet's own Rust test `predict_multisig_address_golden_vector`.
7. A control case runs an honest record through the same flow and it is returned the same way, so the attack case is not accepted by accident.

IMM-WA-CRITICAL-89802

Mobile signing path never validates the endpoint-supplied genesis hash, so a malicious RPC binds signatures to another chain

Report ID	#89802
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/quantus_sdk
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

The genesis hash, spec version and transaction version are signed parameters: they bind a signature to one chain and one runtime. The SDK knows which chains it is willing to sign for. `knownNetworks` lists Planck and Heisenberg by genesis hash under the comment "A payload whose genesis hash is not listed here is rejected" ([quantus_payload_parser.dart#L34-L39](#)), and `QuantusPayloadParser.parsePayload` enforces it, throwing on any genesis hash absent from the map ([#L140-L143](#)). The cold wallet runs that check before it will display or sign anything ([sign_transaction_screen.dart#L44-L52](#)).

The mobile signing path does not go through the parser. `SubstrateService.getExtrinsicPayload` issues five concurrent RPC requests, `_getGenesisHash` among them ([substrate_service.dart#L213-L224](#)), and `_getGenesisHash` asks the same endpoint that will receive the transaction and returns whatever it answers:

```
Future<dynamic> _getGenesisHash() async {
  final result = await _rpcEndpointService.rpcTask((uri) async {
    final provider = Provider.fromUri(uri);
    return await provider.send('chain_getBlockHash', [0]);
  });
  return result.result.replaceAll('0x', '');
}
```

[substrate_service.dart#L365-L371](#)

That value, with the endpoint's `specVersion` and `transactionVersion`, goes straight into the `SigningPayload` ([#L226-L239](#)) and the payload is signed at [#L255](#). `getUnsignedTransactionPayload` repeats the pattern ([#L289-L317](#)).

Nothing between the RPC response and the signature compares any of the three fields against a locally held chain identity. `substrate_service.dart` contains no reference to `knownNetworks`, `parsePayload` or

`specMatchesBundled`: the whitelist ships in the same package and is reached only from the cold-wallet screen. The correct chain is known at build time — `AppConstants.rpcEndpoints` is hardcoded to three Planck hosts ([app_constants.dart#L14-L19](#)) and `bundledSpecVersion` and `bundledTransactionVersion` are pinned constants ([#L64-L65](#)) — and the wallet still signs whatever the endpoint declares.

An attacker controlling one of the three configured endpoints answers `chain_getBlockHash` with `[0]` using Heisenberg's genesis hash `a5aa9e5c...dc8d3b4f` in place of Planck's `4901bf5c...eef65e72`, and answers `state_getRuntimeVersion` with Heisenberg's spec and transaction versions to match, proxying every other request to a genuine node so the wallet behaves normally in all other respects. The user's ML-DSA key then signs a transaction bound to Heisenberg. The send review screen shows recipient, amount, fee and total ([encrypted_send_strategy.dart#L104-L114](#)) and never names the chain, so the user approves an ordinary-looking transfer. The attacker withholds the extrinsic from Planck and submits it to Heisenberg, which accepts it because the signature commits to Heisenberg's genesis hash and runtime versions. One seed derives the same address on every Quantus network, so the account debited is the user's own, on a network they never selected, and the transfer they authorised never occurs. Every transaction signed through `substrate_service.dart` is affected; the cold-wallet path is not. All endpoints are HTTPS, so the precondition is endpoint compromise rather than network interception. This is a malicious interaction with an already-connected wallet that submits a malicious transaction.

Proof of Concept

1. The attacker controls one of the three hardcoded RPC endpoints and proxies all but two requests to a genuine Planck node, so the wallet behaves normally in every other respect.
2. The user initiates a transfer, and `getExtrinsicPayload` fires its five concurrent requests, including `getRuntimeVersion` and `_getGenesisHash`.
3. The attacker answers `chain_getBlockHash [0]` with Heisenberg's genesis hash instead of Planck's, and `state_getRuntimeVersion` with Heisenberg's spec and transaction versions.
4. `_getGenesisHash` strips the `0x` prefix and returns the attacker's value with no comparison; neither `knownNetworks` nor `parsePayload` is reached anywhere on this path.
5. The attacker's genesis hash, spec version and transaction version are placed into the `SigningPayload` unchanged, and the payload is signed with the user's ML-DSA key.
6. The confirmation screen shows only the amount and recipient, never the chain, so the user approves.
7. The attacker withholds the extrinsic from Planck and submits it to Heisenberg, which accepts it; the user's Heisenberg account is debited and the Planck transfer never occurs.

IMM-WA-HIGH-88348

`Utility.if_else` decoding describes its primary branch twice, letting a 213-byte proposal freeze the wallet UI isolate

Report ID	#88348
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Taking down the application/website

Description

`CallDecoder` turns the SCALE-encoded inner call of a multisig proposal into a display tree so that a signer sees every parameter before approving. Its doc comment sets the contract at [call_decoder.dart#L48-L60](#): an exact byte fit, then a `describe` walk visiting each node once. Four of the five recursive wrappers in the file honour that walk, hoisting the recursive result into a local and reusing it. The `Utility.if_else` arm does not.

`describe(main)` runs once at [call_decoder.dart#L577](#) to build the `Primary call` field and again at [#L580](#) to produce `summary`. The same operand, with its entire descendant subtree, is therefore rebuilt at every level: the recurrence $D(n) = 2 \cdot D(n-1) + 2$ resolves to $3 \cdot 2^n - 2$, while the encoded call grows by five bytes per level. At depth 42 the call is 213 bytes with 85 runtime nodes and requires 13,194,139,533,310 `describe` invocations; the runtime regression proposes those exact bytes into an ordinary 2-of-2 multisig, which stores them with status `Active` at 2.080% of `MaxCallSize`.

The application reaches that conversion with no user action beyond launching.

[home_screen.dart#L219-L226](#) mounts `MultisigActivitySection` whenever the active account is a multisig, and [multisig_service.dart#L226-L232](#) routes every open row through `withOnChainCallData`, which reads `decodedCall` on the authoritative storage bytes, synchronously, on the isolate that draws the interface. The active account is persisted and reloaded at launch, so a force-close and relaunch re-enters the same conversion. One valid, cheap, stored proposal is therefore enough to stop the wallet servicing frames or input, taking down the application/website.

Severity note. The project confirmed the duplicate traversal and the exponential expansion as a real defect, but assessed the outcome as a frozen or restarted wallet with no funds at risk. It also noted that the proposal can only come from an existing signer of a multisig the affected user already tracks, that multisig accounts are reached through an advanced menu, and that none of the project's own tools construct an `if_else` payload.

IMM-WA-HIGH-88483

Approve sheet signs multisig bytes it failed to decode, showing amount zero and an empty destination

Report ID	#88483
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Direct theft of user funds

Description

The multisig pallet binds an approval to exact bytes — `approve` rejects anything that is not byte-equal to the stored payload (`src/lib.rs#L779-L781`) — so that signers "can decode and inspect what they are signing instead of trusting an opaque proposal id" (`#L722-L726`). The mobile approve sheet exists to perform that inspection.

Two decoders disagree about the same blob. The pallet validates the inner call with `Decode`, not `DecodeAll`, at both `'propose#L599-L600'` and `'execute#L1125-L1126'`, so a trailing `0x00` is ignored while the full `BoundedVec`, padding included, is stored. The SDK decoder is strict: `CallDecoder.decodeBytes` throws a `FormatException` on any remainder, pinned by the unit test at `'call_decoder_test.dart#L292-L297'`.

The application takes the wrong side of that disagreement. `_withOnChainCallData` treats a decode failure as "not a transfer" rather than as a refusal to sign: `decodedCall` catches the exception and returns null, so a correct indexer amount is overwritten with `0` and an empty recipient. The confirm sheet's gate is `blocked = _submitting || _loadingCallBytes || _callBytesFailed`, and `_callBytesFailed` is set only when the RPC byte load throws, never on a decode throw, so Approve stays enabled (`'multisig_action_confirm_sheet.dart#L185-L195', '#L361-L362'`).

A clean `Balances.transfer_allow_death` encodes to 43 bytes; the same encode plus one `0x00` is 44 bytes and still prefix-decodes to the attacker as destination and the full vault as value. The victim sees amount 0 with no destination, approves, and the app resubmits the stored 44 bytes; the threshold is met and the attacker executes. On a local run against the production `quantus-runtime`, the vault fell from 1000 UNIT to 0 and the attacker netted +998.98 UNIT. This is direct theft of user funds from every non-high-security multisig with threshold at least 2 whose remaining signer uses the official approve sheet.

Severity note. Both declared impacts are Critical. The drain is not unilateral: it completes only once the honest signer approves through the ordinary confirmation flow, which bounds the finding below the Critical class.

IMM-WA-HIGH-90460

Quantus POS latches **Payment Received** on a pre-execution pool notification the payer can still replace

Report ID	#90460
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Merchant POS irreversibly reports an invoice paid after the payer replaces it before execution, enabling goods or services to be released without payment

Description

PosQrScreen is the merchant-facing checkout: it renders a payment QR for a fixed invoice amount and must not report payment until that invoice transfer actually pays. The repository documentation states the integration rule directly — **txWatch_transfer** is not a confirmation.

TxWatchService.watch opens a WebSocket **txWatch_watchAddress** subscription and forwards every matching notification to its **onTransfer** callback (``tx_watch_service.dart#L57-L65``). The node emits that notification when a transaction enters the ready pool, before execution or finality, and it carries no inclusion, dispatch-success or finality field. The POS callback validates nothing else:

It then latches **_paidTransfer = tx**, **_isPaid** drives the app-bar title to **posQrTitlePaymentReceived** and swaps the bottom controls for **Done** and **New Charge** (``pos_qr_screen.dart#L173`, #L188`).

That latch is assigned once and never cleared. The screen registers the transfer with the pending-transaction poller but passes it no callback able to reset the state, and subscribes to no replacement, failure or finality status, so nothing in the screen can withdraw the success it has already reported. The paid branch also calls **_buildPaidContent(...)** at **#L184** without returning it, leaving the QR body on screen; that rendering defect does not undo the paid state, because the title, the completion controls and the cancelled timeout all still change.

The payer therefore signs an ordinary invoice transfer with nonce **0** and no tip, submits it through the normal external path, and waits for the merchant's POS to display **Payment Received**. Before inclusion, the payer submits a different signed transfer with the same signer and nonce and a higher tip. Runtime validation gives both the same pool dependency tag and the replacement the higher priority, so the production pool emits **usurped**, removes the invoice and retains only the replacement. No miner, validator, RPC compromise,

reorg, privileged role or invalid transaction is involved — both are real ML-DSA-65-signed extrinsics submitted as `TransactionSource::External`. The POS remains in the paid state permanently, so a merchant relying on the first-party screen releases goods or services for an invoice that will never execute, and the loss scales with the invoice value accepted.

IMM-WA-HIGH-89806

SDK `RuntimeCall` decoder lacks the chain's Vesting index, leaving an undecodable multisig approval enabled

Report ID	#89806
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

An actionable multisig review must decode every nested authoritative call before the signer commits; the SDK's own note requires that signers "warn loudly rather than present a decode they cannot vouch for" ([`app\\_constants.dart#L60-L65`](#)).

The frozen chain registers `Vesting` at `RuntimeCall` pallet index 22 ([`src/lib.rs#L286-L287`](#)), and the SDK's generated `$RuntimeCallCodec.decode` has no case for it. Any canonical call containing `Vesting` therefore throws: `MultisigProposal._decodeCall` catches the exception and returns null ([`multisig\\_proposal.dart#L268-L274`](#)), and `_withOnChainCallData` clears the recipient and amount.

The confirmation screen never converts that semantic failure into a blocking state: its gate is `blocked = _submitting || _loadingCallBytes || _callBytesFailed`, and `_callBytesFailed` is set only in the `catch` around the byte loader ([`multisig\\_action\\_confirm\\_sheet.dart#L158-L181`](#)), so successful retrieval of bytes that cannot be reviewed is treated as sufficient to sign.

A 49-byte canonical `Utility.force_batch([Vesting.claim(u64::MAX), Balances.transfer_all(attacker, keep_alive)])` is accepted and stored by an ordinary non-high-security 2-of-2 multisig. The victim's sheet fails to decode those bytes and renders only the opaque top-level `Force batch | Utility` text — the attacker address, the amount and both nested calls are absent, and no decode warning is shown. `Yes, Approve` remains enabled, the resubmitted bytes match storage exactly, and approvals reach 2 of 2. On execution `force_batch` continues past the failing `Vesting.claim` and `transfer_all` moves 999.999 QUAN of a 1,000 QUAN multisig to the attacker. This is a malicious interaction with an already-connected wallet.

Severity note. Both declared impacts are Critical. The report's own stated limit is that the victim's approval is required to cross a genuine threshold, and that the screen shows an opaque wrapper with missing children rather than a benign decoy recipient or a decoded safe transfer.

IMM-WA-HIGH-89095

Android recovery-phrase clipboard expiry relies on an in-process timer that dies with the app

Report ID	#89095
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-recovery/mobile-app
Impact	Improperly disclosing confidential user information, such as: Email address · Phone number · Physical address, etc.

Description

`SecureClipboardService` bounds how long a sensitive value survives on the system clipboard: its doc comment promises the text is "cleared after [ttl]", preferring "OS-enforced expiration", with `defaultTtl` at three minutes (``secure_clipboard_service.dart#L27-L37``). The 24-word recovery phrase is sent through it by ``recovery_phrase_body.dart#L59-L63``.

`copyWithExpiry` takes the native, OS-enforced path only on iOS, where `AppDelegate` sets `UIPasteboard` items that expire "even while the app is suspended or terminated" (``AppDelegate.swift#L28-L72``). The repository contains no `app.quantus/secure_clipboard` handler on the Android side, so every Android copy falls through to a plain `Clipboard.setData` and an in-process Dart `Timer`. When Android kills the backgrounded wallet under ordinary lifecycle pressure, that timer disappears while the phrase remains in the system clipboard, and a fresh launch constructs a new service instance with no persisted deadline. The Android copy also omits `ClipDescription.EXTRA_IS_SENSITIVE`, so the phrase is not concealed in clipboard previews.

From Android 10 onward a clipboard reader must be the foreground application or the default IME, so the exposure is realised by a separate installed application opened normally after the advertised deadline. The PoC measures that boundary: in the process-death arm a native reader foregrounded roughly 188 seconds after the copy recovered the 24-word canary and reported `sensitive_extra=false`, while the live-process control at the same boundary received an empty value — isolating process termination, not timing, as the cause. On the tested Android 16 device the system auto-clear timeout was 3,600,000 ms, so the OS did not enforce the promised 180-second bound either. Without this defect, an application first foregrounded after three minutes receives an empty clipboard; with it, that application receives the wallet's recovery phrase, improperly disclosing confidential user information.

Report ID	#88914
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Direct theft of user funds

`DecodedCallView` renders every native-token `AmountField` through the general transaction display provider, which watches `isBalanceHiddenProvider`: with the global privacy setting active every amount collapses to the literal `-----` (``currency_display_provider.dart#L242``, ``#L283-L310``). Separately, `CallDecoder` assigns `utility.if_else` the summary of its primary branch alone, so the confirmation headline is derived only from `main`.

[illegible]

Severity note. Both declared impacts are Critical. The outcome depends on the victim having the global Hide balances setting enabled when entering the authorization flow and then approving, and the report itself identifies that user-interaction dependency as the reason it does not claim Critical.

IMM-WA-MED-88274

`Utility.batch` decoding drops the nested transfer summary, hiding the amount on the multisig approval sheet

Report ID	#88274
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

The multisig approval sheet is the screen that asks a co-signer to authorise a stored proposal, and its renderer states that batches are among the proposal shapes it must render correctly rather than assume to be transfers ([decoded_call_view.dart#L9-L14](#)). `CallDecoder._transfer` attaches a `TransferSummary` to every decoded transfer ([call_decoder.dart#L140-L147](#)), and the sheet builds its headline amount from that summary through `DecodedCallHeadline.of` with `fmt.formatBalance` ([multisig_action_confirm_sheet.dart#L354-L360](#)). Every other wrapper in the same `switch` lifts its inner call's summary upward: `as_derivative`, `with_weight` and `if_else` all pass it ([call_decoder.dart#L546-L581](#)), as do `dispatch_as` and `Recovery.as_recovered`. `_batch`, which serves `batch`, `batch_all` and `force_batch`, does not:

The nested amount then survives only as an `AmountField`, which renders through `txAmountDisplayProvider` ([decoded_call_view.dart#L66-L69](#)); that provider watches `isBalanceHiddenProvider` and replaces both amounts with `-----` ([currency_display_provider.dart#L374-L380](#)). With Hide balance on, a one-item `Utility.batch` proposal therefore states its amount nowhere on the sheet, while the same transfer unwrapped still states it once in the headline. Confirmation stays enabled — `blocked` covers only submission, byte loading and byte-load failure ([multisig_action_confirm_sheet.dart#L361-L362](#)) — and forwards the authoritative bytes unchanged. The report's 46-byte fixture decodes to a transfer of 987654321000000 base units, 987.654321 tokens at the chain's 12 decimals, and the runtime regression asserts that exact delta on the destination once a second registered signer approves. The second signature is real, but a principal transaction parameter is absent from the screen that requests it, which is a malicious interaction with an already-connected wallet.

Severity note. The exposure is bounded by conditions the report states: Hide balance is opt-in and off by default ([settings_service.dart#L386-L388](#)), the proposer must already be a registered signer of the multisig and below its threshold alone, the wrapper and destination remain visible on the sheet, no signature or threshold check is bypassed, and the finding does not apply at threshold 1 because no second approval is solicited.

IMM-WA-MED-89188

UTXO pager orders only by block height, so a page boundary drops a same-height transfer from the claim list

Report ID	#89188
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/quantus_sdk
Impact	Temporary freezing of funds for at least 24 hour

Description

Wormhole notes are keyless, so the only spend path is the inbound `transfer` list the SDK assembles: a row that never enters that list cannot be turned into a UTXO and proved out. `WormholeUtxoService` pages those rows 300 at a time ([wormhole_utxo_service.dart#L127-L130](#)) with `order_by: {block: {height: asc}}` and nothing else ([#L296-L313](#)), and `fetchAllTransfers` walks them with a plain `offset += _transferPageSize` ([#L366-L395](#)). Block height is not unique, however, and `LIMIT/OFFSET` over a non-unique ordering does not partition the table: when three or more transfers share a height sitting on a 300-row boundary, one sibling is returned twice and another is omitted. The walk then writes a disk cache at `cachedUpToBlock = chainHeight - _reorgDepth` and subsequently queries only `height_gt` that cutoff ([#L507](#), [#L555-L563](#)), so the skipped finalized height is never asked for again, and clearing the cache repeats the same height-only walk and loses the same row. `claimRewards` reaches `getUnspentTransfers` without clearing anything ([wormhole_send_service.dart#L250-L258](#)). Against local Postgres running the query the shipped service issues, 2439 rows returned but only 2438 were unique, and a 2.59 QUAN note was absent from the spendable set. A phone or miner-app holder cannot change the ordering, so the note stays unspendable until an app or SDK release ships — a temporary freeze of those funds measured in days, not hours.

IMM-WA-MED-88786

Point-of-sale screen promotes a pre-execution pool notification to paid, so a rolled-back `batch_all` reads as payment received

Report ID	#88786
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	[GAP: impact — not declared in the export]

Description

The node's integration document for `txWatch_transfer` sets the contract for this client in its own words: never release goods, mark an invoice paid, or credit a balance on a notification alone, because anyone can submit a `batch_all` holding one real-looking transfer and one transfer that must fail (`docs/rpc_additions/transaction_pool_listener.md`, chain repository). The notifier matches that description: it takes the extrinsic from the ready pool and calls `extract_all_transfers`, which descends into `batch`, `batch_all` and `force_batch`, broadcasting before any execution and without consulting a dispatch event (`node/src/txwatch.rs`). In the wallet, `PosQrScreen`'s `onTransfer` callback compares only the announced amount against the invoice, records a `PendingTransactionEvent`, calls `startPolling(pendingTx)` and assigns `_paidTransfer` (`pos_qr_screen.dart#L75-L105`). `startPolling` accepts an optional `onFound` confirmation callback (`pending_transaction_polling_service.dart#L23`), but the screen supplies none, so nothing gates the paid state on the poller's result; `_isPaid` is `_paidTransfer != null` (`#L48`), and `_paidTransfer` is assigned at one place and never reset, so there is no revocation path. `_isPaid` drives the app bar title to `posQrTitlePaymentReceived` and swaps the bottom actions to Done / New Charge (`#L172-L189`). A customer who wraps the invoice transfer in `Utility.batch_all` beside a child that must fail therefore reaches that state, and the runtime rolls the whole batch back: the PoC asserts the merchant's balance is exactly unchanged. The merchant releases goods against a confirmation the project's own documentation prohibits acting on, at the cost of a failed extrinsic's fee.

IMM-WA-MED-88221

Unvalidated `amount` in the `/pay` deep link freezes the wallet on unbounded `BigInt` materialisation

Report ID	#88221
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Taking down the application/website

Description

The wallet registers `https://www.quantus.com/pay` as an auto-verified Android App Link (`AndroidManifest.xml#L44-L52`) and as an iOS Universal Link via `applinks:www.quantus.com`, so an ordinary link on any page opens it. `DeepLinkService` fails closed on the recipient and on the recipient only (`deep_link_service.dart#L59-L68`), while `PaymentIntent.tryParseUrl` carries `amount` as a raw string and tests it for emptiness alone (`route_intent_providers.dart#L56-L63`). The home screen pushes `InputAmountScreen(initialAmount: ..., isPayMode: true)` (`home_screen.dart#L135-L143`), whose `initState` parses that string synchronously on the UI isolate through `parseWireAmount` (`input_amount_screen.dart#L74-L87`). Normalisation strips grouping separators and swaps the decimal mark, but it does not reject or bound exponent notation (`locale_number_config.dart#L65-L82`), so `1e100000000` reaches `parseAmount`, which multiplies the lazily held value by the chain's `10^12` scale factor and calls `toBigInt()` (`number_formatting_service.dart#L132-L148`). That call materialises the full integer — 33,219,321 bits for that input — and the surrounding `try/catch` catches exceptions, not a hang. The harness measured 42–53 s of blocking on a desktop arm64 core against a 5 ms control, the cost grows superlinearly with an exponent nothing bounds, and the project's own device reproduction ended in the operating system killing the unresponsive app after roughly six minutes. Because the parse runs on the UI isolate inside `initState`, the wallet accepts no input for the duration, taking down the application.

Severity note. The denial is per tap rather than persistent: the `PaymentIntent` lives in an in-memory provider that is nulled before the send flow is pushed (`home_screen.dart#L128`), so relaunching from the icon yields a working wallet, no funds are stolen or frozen on chain, and no key material sits on the affected path.

IMM-WA-MED-89478

Balance-privacy display blanks the transfer amount on the multisig approval sheet while Approve stays enabled

Report ID	#89478
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

The multisig approval sheet is the last screen a co-signer sees before signing, and it decodes the authoritative stored call bytes correctly: the wrapper `Utility · batch`, the inner `Balances · transfer_allow_death` and the full destination address all render. Each parameter is drawn by `DecodedCallView`, and an `AmountField` takes its value from `txAmountDisplayProvider` ([decoded_call_view.dart#L66-L69](#)) — the same provider the portfolio uses, which watches `isBalanceHiddenProvider` and rewrites both the primary and secondary amount to the placeholder `-----` ([currency_display_provider.dart#L374-L380](#)). Balance privacy is a portfolio-display preference, however, not an authorisation to omit a transaction parameter from a signing confirmation: the sheet offers no per-field reveal and no warning, and it leaves Approve enabled, because its `blocked` predicate covers only submission, byte loading and byte-load failure ([multisig_action_confirm_sheet.dart#L361-L362](#)). A co-signer of an ordinary funded 2-of-2 multisig who has enabled the toggle is therefore asked to approve `Utility.batch([Balances.transfer_allow_death(attacker, 1,000,000 QUAN)])` with `Amount: -----` on screen. The proof of concept taps Approve, signs the exact bytes with the production ML-DSA key, and the runtime accepts the byte-identical extrinsic and moves 1,000,000 QUAN out of the multisig to the proposer. The approval that meets the threshold is genuine; the amount it authorises was never displayed, so the wallet submits a malicious transaction on its holder's behalf.

Severity note. The report's own scenario bounds the exposure: the proposer must already be a registered co-signer of a funded multisig and cannot reach the threshold alone, the victim must have turned on the balance-privacy toggle that the attacker cannot change, and the wrapper, the inner call name and the destination address all remain visible on the sheet.

IMM-WA-LOW-88305

`predictMultisigAddress` omits the pallet's signer deduplication and returns an address the chain never creates

Report ID	#88305
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/quantus_sdk
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

`pallet-multisig` derives a multisig account from the sorted **and deduplicated** signer list and documents that determinism as precisely what lets callers pre-compute the address, and the SDK's `predictMultisigAddress` is that pre-computation. Its Rust derivation only sorts ([multisig.rs#L8-L25](#)) and the shared Dart guard checks only cardinality ([multisig_service.dart#L216-L223](#)), so a list carrying a repeated signer passes both the prediction and `buildCreateMultisigCall` unchanged. The SDK then returns one address while `create_multisig` succeeds at another. `resolveMultisigCreationParams` tests takenness only against addresses the chain will never create ([#L87-L113](#)), and the predicted value becomes `MultisigAccount.accountId` ([multisig_submission_service.dart#L114-L126](#)) — the identity a caller stores, displays and funds. Value sent there lands on a hash-derived account with no key and no multisig record, so the address the connected wallet operates on has been substituted for one that cannot exist.

Severity note. No adversary and no beneficiary are involved. The shipped wallet cannot produce the divergent input, because its creation screen rejects a repeated signer before any prediction is made ([add_multisig_screen.dart#L118-L121](#)), and it never presents the predicted address as a funding target; the exposure is `quantus_sdk` as a library a third-party integrator calls directly.

IMM-WA-LOW-88263

Late check-phrase lookup overwrites the current recipient's phrase, so the review screen vouches for another address

Report ID	#88263
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/mobile-app
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions

Description

The wallet's defence against a substituted recipient is a human-readable check phrase, computed in a background isolate behind a per-address cache that returns a hit before any isolate work ([human_readable_checksum_service.dart#L66-L99](#)). Inside one method of `SelectRecipientScreen` the self-send probe drops a late answer when the field no longer holds the address it was started for; the check-phrase lookup thirteen lines below writes its result unconditionally ([select_recipient_screen.dart#L122-L140](#)). A slow miss for a pasted address therefore lands after a fast hit for a recent one and overwrites the kept address's phrase. The value is propagated, not re-derived: `InputAmountScreen` accepts a phrase it was handed ([input_amount_screen.dart#L88-L94](#)), the review screen renders it beside the recipient ([review_send_screen.dart#L199-L205](#)), and the hardware-signing session shows it as its third detail line ([regular_send_strategy.dart#L129-L138](#)). The control meant to catch a substituted address instead vouches for one the user is not paying, on the last screen before the signature.

Severity note. There is no attacker angle. Both address changes are made by the user, no remote input reaches this screen, the recents list is fed only by the user's own outgoing transfers, and the full address is displayed next to the phrase; reaching the state requires a specific and fast sequence of taps that is not a common way to use the screen.

IMM-WA-INSIGHT-88382

Wallet mnemonic stored with the default iOS Keychain accessibility class

Report ID	#88382
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-quantus-apps/tree/audit-comp-ready/quantus_sdk
Impact	—

Description

`SettingsService` builds its only `FlutterSecureStorage` with no `IOSOptions` ([settings_service.dart#L16](#)) and writes the BIP39 mnemonic through it ([#L349-L358](#)), so on iOS the item takes the library default `kSecAttrAccessibleWhenUnlocked` rather than a `ThisDeviceOnly` class; no accessibility option appears anywhere in the two in-scope Dart trees. Such items are copied into an encrypted device backup and restored onto other hardware, so the seed from which every account key derives is neither backup-excluded nor device-bound.