



IMMUNEFI

AUDIT COMPETITION



ImmuneFi /



Firelight

DATE

Aug. 25, 2026

TOP AUDITORS

Griffin
Tradi3
anbug

REPORT BY

ImmuneFi

01

Overview

02

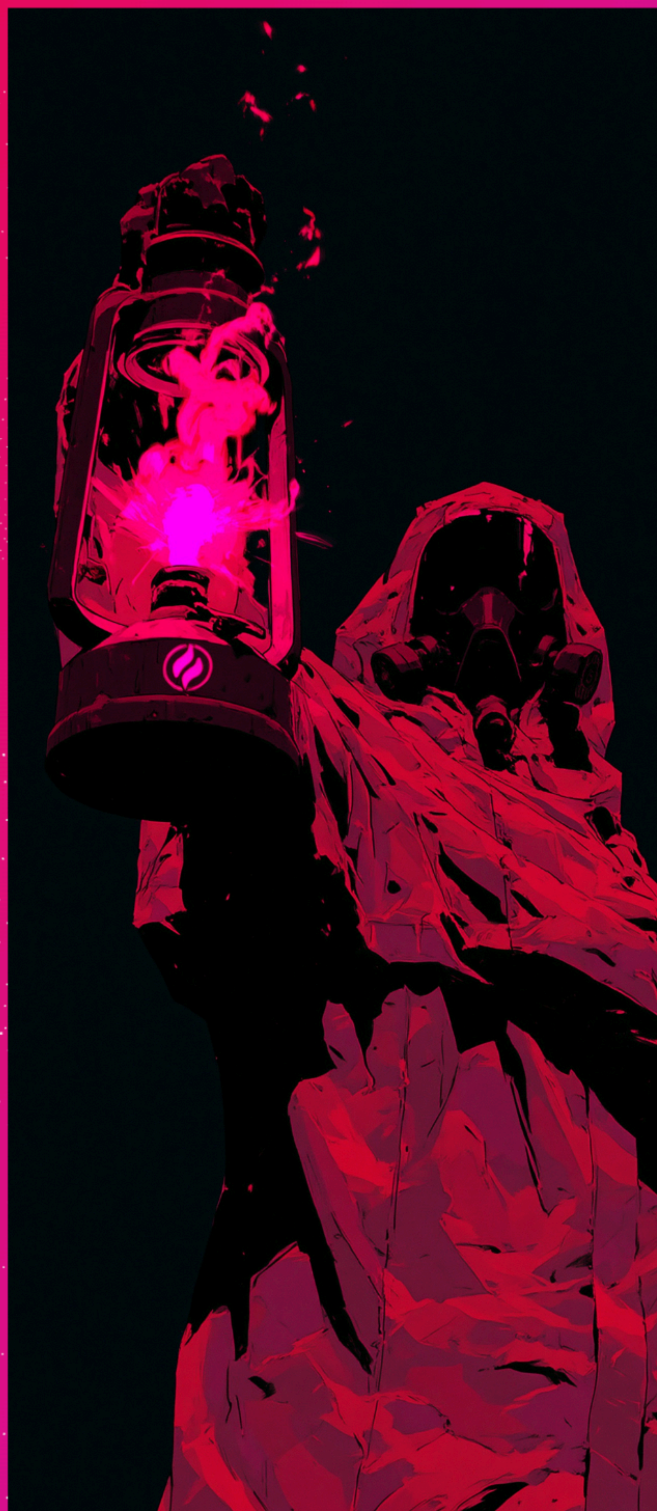
Scope of assets

03

Summary

04

Findings



Contents

About Immunefi	3
Immunefi Audit Competitions	4
Terminology	5
Executive Summary	6
Findings — Smart Contract	8

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 Firelight 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.

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 band below is the one that applies to the assets in scope of this competition.

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

Smart Contract

Severity	In-scope impacts
Critical	• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties • Permanent freezing of funds • Permanent freezing of NFTs • Unauthorized minting of NFTs • Predictable or manipulable RNG that results in abuse of the principal or NFT • Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content) • Protocol insolvency
High	• Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds for at least 24 hour • Temporary freezing of NFTs for at least 24 hour
Medium	• Temporary freezing of funds for at least 1 hour • Temporary freezing of NFTs for at least 1 hour • Smart contract unable to operate due to lack of token funds • Block stuffing • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption
Low	• Contract fails to deliver promised returns, but doesn't lose value

Executive Summary

Over the course of 13 days in August 12 – August 25, 2026, Firelight ran an audit competition on Immunefi covering 16 contracts across Smart Contract. 189 reports were confirmed, resolving into **8 unique findings**: 4 High, 2 Medium and 2 Low, plus 2 Insights.

Summary

Name	Firelight
Competition Type	Audit Competition
Competition Period	August 12 – August 25, 2026
Reward Pool	\$20,000 USD
Languages	Solidity
Scope	3,044 nSLOC across 16 contracts
Repository	immunefi-team/audit-comp-firelight
Commit	42f9ea5e43d88b35197b901acc2a8c24b314fa62
PoC Requirement	Runnable, required at every severity
Confirmed Reports	189
Unique Findings	8

Assets in Scope

Contract
contracts/core/FirelightVault.sol
contracts/core/FirelightVaultStorage.sol
contracts/core/CoverOrderAllocator.sol
contracts/core/CoverNFT.sol
contracts/core/IncidentManager.sol
contracts/core/VaultRewardDistributor.sol
contracts/core/interfaces/IFirelightVault.sol
contracts/core/interfaces/ICoverOrderAllocator.sol
contracts/core/interfaces/IIncidentManager.sol
contracts/core/interfaces/IAggregatorV3.sol
contracts/core/lib/Checkpoints.sol
contracts/core/lib/Decimals.sol
contracts/core/lib/PriceFeed.sol
contracts/legacy/FirelightVaultPredeposits.sol

Contract
contracts/legacy/FirelightVaultStoragePredeposits.sol
contracts/oracle/FtsoChainlinkAdapter.sol

Issues Found

Severity	Smart Contract	Total
Critical	0	0
High	4	4
Medium	2	2
Low	2	2
Total	8	8
Insight	2	2

Findings index – Smart Contract

Finding ID	Severity	Report Name
IMM-SC-HIGH-88287	High	Inclusive period-start checkpoint lets a boundary <code>withdraw()</code> zero the payout cap while the assets stay payable
IMM-SC-HIGH-88214	High	Cover capacity for period <code>P+1</code> counts collateral still liable for an unpaid period- <code>P</code> claim
IMM-SC-HIGH-88231	High	<code>rescueWithdrawFromBlocklisted</code> marks a future withdrawal bucket claimed, permanently freezing later requests into it
IMM-SC-HIGH-88525	High	<code>addPeriodConfiguration</code> accepts an unbounded period duration, permanently stalling withdrawal progression
IMM-SC-MED-88366	Medium	<code>approveCurrentAssessment</code> binds only the incident id, so a replaced assessment round inherits approval signed for another
IMM-SC-MED-88330	Medium	Post-slash zero-assets state inverts the ERC4626 virtual offset, letting a 1-wei deposit mint a share-supply majority
IMM-SC-LOW-88234	Low	<code>maxMint</code> and <code>maxDeposit</code> advertise amounts that <code>mint</code> and <code>deposit</code> reject
IMM-SC-LOW-88269	Low	Missing zero-assets guard in <code>redeem</code> burns shares for nothing when the share price is below 1
IMM-SC-INSIGHT-88930	Insight	NatSpec across <code>FirelightVault</code> , <code>ICoverOrderAllocator</code> and <code>IIncidentManager</code> describes behaviour the implementations do not have
IMM-SC-INSIGHT-88963	Insight	<code>CapacityConfig</code> field order leaves <code>divergenceToleranceBps</code> in a storage slot of its own

Findings – Smart Contract

IMM-SC-HIGH-88287

Inclusive period-start checkpoint lets a boundary `withdraw()` zero the payout cap while the assets stay payable

Report ID	#88287
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Protocol insolvency

Description

`FirelightVault` is the ERC4626 vault whose staked assets back the cover sold by `CoverOrderAllocator` and are slashed by `IncidentManager` when an incident is assessed. Exposure is organised in periods, and one quantity serves both sides of that exposure: `totalAssetsAt(currentPeriodStart())` bounds how much cover may be sold for a period in [CoverOrderAllocator.sol#L704-L719](#), and the same snapshot caps how much may be paid out of that period. `payout()` states the accompanying invariant in its own comments: withdrawals requested during the capture period are assigned to `capturePeriod + 1` and "remain payable while payout is allowed", because the tokens are still held by the vault during the cooldown.

A withdrawal performs two writes. `_requestWithdraw()` books the assets into `withdrawAssets[currentPeriod() + 1]` and burns the shares at [FirelightVault.sol#L1008-L1019](#). `withdraw()` then calls `_logTrace()`, which pushes the post-withdrawal total under `Time.timestamp()` at [FirelightVault.sol#L1044-L1055](#). `totalAssetsAt()` resolves that trace through `Trace256.upperLookupRecent()`, which returns the last checkpoint with a key lower **or equal** to the search key ([Checkpoints.sol#L217-L224](#)).

At `block.timestamp == periodStart`, `currentPeriod()` has already advanced, so the request is filed against the new period and stays payable for it. However, the checkpoint `_logTrace()` writes carries exactly the key `periodStart`, and the inclusive lookup admits it, so `totalAssetsAt(periodStart)` returns the post-withdrawal figure for a snapshot that is meant to bound pre-boundary exposure. `payout()` reads both quantities and takes the minimum:

In the PoC a staker deposits 1,000 FXRP and withdraws at the exact first timestamp of period 1. The vault still holds 1,000 FXRP and `withdrawAssets[2]` records all of it, but `totalAssetsAt(periodStart)` returns 0. The matcher then commits the capacity the chain itself reports — a funded \$100 first-loss buffer at the

configured 2x ratio, so exactly \$200 — and settles \$200 of cover. A valid \$200 assessment transfers the full \$100 buffer and asks the vault for the remaining 100 FXRP, which [FirelightVault.sol#L926-L927](#) caps to `min(100, min(0, 1000)) == 0`. The incident closes with `vaultPaidAmount == 0` and the staker later claims all 1,000 FXRP; the same run with the withdrawal one second later pays the 100 FXRP and leaves the staker 900. A settled cover obligation closes \$100 short while the collateral that answers for it remains in the vault, which is protocol insolvency.

IMM-SC-HIGH-88214

Cover capacity for period $P+1$ counts collateral still liable for an unpaid period- P claim

Report ID	#88214
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Protocol insolvency

Description

`CoverOrderAllocator` bounds how much cover Firelight may sell in a period, and it is the chain rather than the off-chain matcher that enforces that bound: `_validateMatchingCapacity()` recomputes the ceiling and reverts `MatchingCapacityOverflow` on any `matchingCapacity` above it ([CoverOrderAllocator.sol#L394-L403](#)). `MIN_CAR_BPS`, `MAX_LEVERAGE_FACTOR` and `divergenceToleranceBps` exist so that the capital-adequacy ratio is a protocol constraint and not a matcher convention.

`_computeAvailableCapacity()` values that collateral as the live first-loss buffer balance plus the vault's period-start staked-asset snapshot, then applies the leverage ratio:

The payout window for a capture period P spans P and $P+1$ ([FirelightVault.sol#L1130-L1132](#)), so a period- P claim is legitimately paid during $P+1$. `FirelightVault` recognises that entanglement on the deposit path: `_hasActiveIncident()` blocks deposits while an incident is active in the current or the previous period, and its comment gives the reason — "otherwise fresh deposits would back unresolved exposure" ([FirelightVault.sol#L1112-L1118](#)).

The capacity path carries no such gate. Neither collateral term above is reduced by exposure already owed, `CoverOrderAllocator` contains no reference to `hasActiveIncident`, and neither `commitAllocation()` nor `_settleCoverOrder()` reads incident state. The documented off-chain formula the matcher is told to implement has no term for unresolved incidents either ([ICoverOrderAllocator.sol#L40-L41](#)). The same collateral therefore backs two periods of cover at once.

The end-to-end PoC runs the real vault, allocator and incident manager with \$200,000 in the buffer and \$1,000,000 staked, a 2x multiplier (`effectiveLeverage` 24000 over `minCAR` 12000) and `divergenceToleranceBps = 0`, the most conservative legal configuration. Period P settles \$1,000,000 of cover and an incident is created, assessed and left under evaluation. In $P+1$ the buffer balance is unchanged and `isPeriodInPayoutWindow(P)` is true, yet the chain grants the identical capacity and a second \$1,000,000 settles: \$2,000,000 of cover against \$1,200,000 of collateral. The period- P claim then pays

\$200,000 from the buffer and \$800,000 from the vault, leaving \$200,000 behind \$1,000,000 of still-live **P+1** cover — 5x against a 2x cap, and \$800,000 of unbacked liability if that cover claims in full. Committing the same capacity figure in **P+2** reverts **MatchingCapacityOverflow**, the number the chain accepted one period earlier on the same collateral. Cover is sold against collateral already committed to an earlier period's claims, and the condition recurs at every period boundary where an incident is unresolved, which is the normal state during an assessment round: protocol insolvency.

IMM-SC-HIGH-88231

`rescueWithdrawFromBlocklisted` marks a future withdrawal bucket claimed, permanently freezing later requests into it

Report ID	#88231
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immuneFi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Permanent freezing of funds

Description

`FirelightVault` settles withdrawals by period bucket: a request is booked against a future period and the assets are released once that period has ended. `isWithdrawClaimed[period][account]` is the terminal marker that a bucket is settled for an account, and `claimWithdraw()` rejects any period already flagged ([FirelightVault.sol#L788-L811](#)). `rescueWithdrawFromBlocklisted()` lets `RESCUER_ROLE` move a blocklisted account's pending withdrawal to a safe address and marks the source as settled.

`_requestWithdraw()` always assigns `currentPeriod() + 1` ([FirelightVault.sol#L1008-L1019](#)), so a pending bucket is by construction a period that has not started and is still open to further requests. The rescue loop credits `to`, zeroes `from`, and sets the flag:

The loop does not require `periods[i]` to have elapsed, and `_requestWithdraw()` never consults `isWithdrawClaimed`. Rescuing a pending bucket therefore writes a terminal flag onto a period that still accepts entitlement, and a later withdrawal by the same account within the same period lands in that flagged bucket and burns the shares at `update(owner, address(0), shares)`.

`isWithdrawClaimed` is written in exactly two places — `claimWithdraw()` at line 806 and the rescue at line 870 — and is never cleared: the contract has no setter, no administrative override and no `delete`. Once the bucket's period ends, `claimWithdraw()` reverts `AlreadyClaimedPeriod` for that account forever, and re-blocklisting and re-rescuing reverts on the same flag at the `from` guard, so the intended remediation path is self-blocking. The assets remain counted in `pendingWithdrawAssets`, and because `totalAssets()` returns `super.totalAssets() - pendingWithdrawAssets` ([FirelightVault.sol#L427-L429](#)) they are also excluded from every other depositor's net asset value: the tokens sit in the vault owned by nobody. The PoC reaches this state entirely inside a single period through ordinary role actions — withdraw, blocklist, rescue, un-blocklist, withdraw again — with no actor exceeding its documented function and no period-number movement involved. The stranded account holds neither shares nor assets, no role including `DEFAULT_ADMIN_ROLE` can clear the flag, and pausing has no effect on it, so only a proxy upgrade recovers the balance. This is permanent freezing of funds.

IMM-SC-HIGH-88525

`addPeriodConfiguration` accepts an unbounded period duration, permanently stalling withdrawal progression

Report ID	#88525
Severity	High
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Permanent freezing of funds

Description

`FirelightVault` settles withdrawals on period boundaries. `_requestWithdraw()` books each request into `currentPeriod() + 1` and `claimWithdraw()` refuses any period that has not fully elapsed — `if (period >= currentPeriod()) revert InvalidPeriod();` at [FirelightVault.sol#L791](#) — so the length of a period is the length of every claim horizon behind it. Period cadence lives in `periodConfigurations`, which `PERIOD_CONFIGURATION_UPDATE_ROLE` extends through `addPeriodConfiguration()` at [FirelightVault.sol#L542-L544](#).

Validation applies a floor and a granularity check, then constrains only the epoch:

`SMALLEST_PERIOD_DURATION` is one day, but no upper bound is applied to `newDuration` and nothing computes `epoch + duration` at schedule time. One call therefore installs a multi-year period, or a duration close to the `uint48` maximum. The escape routes are closed by the same function: the minimum notice is one boundary but the role chooses the scheduling moment, a pending configuration cannot be replaced because `CurrentPeriodConfigurationNotLast` rejects the second call, `periodConfigurations` is push-only with no removal function, and once the oversized entry is active the earliest epoch accepted is `nextPeriodEnd()` — two full oversized durations away.

Every withdrawal booked into or after the oversized period stays unclaimable until that period ends, while deposits and share transfers continue, so capital enters the vault but cannot leave. With a near-maximum duration the freeze has no end: `nextPeriodEnd()` adds the duration to `currentPeriodEnd()` in checked `uint48` arithmetic and panics ([FirelightVault.sol#L360-L363](#)), and because `_addPeriodConfiguration()` itself calls `nextPeriodEnd()`, no corrective configuration can ever be installed: the no-cancel guard blocks correction while the schedule is pending and the panic blocks it once the schedule is active, with no window between the two. `currentPeriodEnd()` panics on the same arithmetic, which also reverts `commitAllocation()`, since it compares `graceExpiresAt` against `$.vault.currentPeriodEnd()` at [CoverOrderAllocator.sol#L346](#), halting cover sales as well. The report's fork PoC upgrades the live Flare proxy to this implementation at block 67321080, preserving 57.36 million FXRP of vault assets, and freezes a real holder's withdrawal of roughly

898,900 FXRP for ten years in the finite case; in the overflow case claims and corrections fail identically one hundred years later. Affected staker principal cannot become claimable under this implementation, and recovery requires a proxy upgrade outside the vault's own role surface. This is permanent freezing of funds.

IMM-SC-MED-88366

`approveCurrentAssessment` binds only the incident id, so a replaced assessment round inherits approval signed for another

Report ID	#88366
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/IncidentManager.sol
Impact	Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)

Description

`IncidentManager` splits the claim flow between a `CURATOR_ROLE` that proposes a loss schedule and a separate `ASSESSMENT_APPROVER_ROLE` that validates it, and [IncidentManager.sol#L403-L418](#) takes only an incident id. It resolves the round to approve at execution time, through `_setCurrentAssessmentStatus`, from mutable storage:

That value moves over an incident's life. `cancelCurrentAssessment` cancels a `DRAFT` or `UNDER_EVALUATION` round ([IncidentManager.sol#L386-L400](#)), and the next `addAssessmentLosses` call increments `incident.currentAssessmentRoundId` to open a replacement ([IncidentManager.sol#L313-L326](#)).

Nothing binds an approval to the round that was reviewed: the approver supplies neither an expected round id nor a hash of the losses it carries. A Safe signature set collected while round 1 is current stays executable after round 2 replaces it, because the curator's calls consume no Safe nonce and the signed Safe transaction hash is unchanged.

A curator therefore has a small schedule reviewed and signed, cancels it, and submits a larger replacement that the pending signatures approve and close. The PoC replaces a reviewed 100-unit round with a 1,000-unit round, moving 900 units of backing beyond the reviewed schedule — 150 first-loss-buffer units and 750 vault units — and cutting a separate staker's withdrawable assets from 1,000 to 250, while every transfer lands on the configured payout receiver. This is griefing: no profit motive for the curator, but damage to the users and the protocol.

IMM-SC-MED-88330

Post-slash zero-assets state inverts the ERC4626 virtual offset, letting a 1-wei deposit mint a share-supply majority

Report ID	#88330
Severity	Medium
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Theft of unclaimed yield

Description

FirelightVault is an ERC4626 vault whose conversion math uses the OpenZeppelin virtual-offset defence, which holds only while `totalSupply()` is zero whenever `totalAssets()` is zero ([FirelightVault.sol#L1058-L1065](#)):

payout breaks that condition by design. It settles an incident claim by pushing a reduced total to the assets trace and burns no shares, and it is capped at `totalAssetsAt(capturePeriodStart)`, the full period-start balance ([FirelightVault.sol#L892-L966](#)). A full slash leaves `totalAssets() == 0` with every pre-slash share outstanding, and with `totAssets` zero the formula collapses to `assets * (totalSupply + 1)`.

Deposits reopen once the incident resolves, and the next deposit is unprivileged. The PoC slashes a 20,000-token victim position to zero, after which a 1-wei deposit mints 20,000,000,001 shares against the victim's 20,000,000,000 — a supply majority for one wei — and a subsequent 10,000-token reward distribution pays the attacker 4,999.99999 tokens; roughly 200 wei captures about 99%. Every premium and incentive later forwarded to the vault is divided on that distorted supply, so the dust depositor takes theft of unclaimed yield from the stakers who recapitalize it.

IMM-SC-LOW-88234

`maxMint` and `maxDeposit` advertise amounts that `mint` and `deposit` reject

Report ID	#88234
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Contract fails to deliver promised returns, but doesn't lose value

Description

`FirelightVault` publishes the full ERC-4626 selector set — its interface inherits `IERC4626` ([IFirelightVault.sol#L22](#)) — and the standard requires `maxMint` and `maxDeposit` to return amounts the matching call will not revert on. `maxMint` derives its ceiling from `convertToShares(depositLimit)`, which floors, while `mint` enforces an asset ceiling and `previewMint` ceils ([FirelightVault.sol#L385-L393](#)); `maxDeposit` returns the remaining asset headroom without checking it mints at least one share, which `deposit` rejects ([FirelightVault.sol#L371-L378](#)). Above a 1.0 share price — the steady state once `VaultRewardDistributor` forwards assets without minting shares — the PoC has `maxMint` return 1 while `previewMint(1)` costs 2 and `mint(1)` reverts `DepositLimitExceeded`, and `maxDeposit` return 1 while `previewDeposit(1)` is 0. Integrators following the canonical pattern submit guaranteed-failing transactions and waste gas: the contract fails to deliver promised returns, but doesn't lose value.

IMM-SC-LOW-88269

Missing zero-assets guard in `redeem` burns shares for nothing when the share price is below 1

Report ID	#88269
Severity	Low
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/FirelightVault.sol
Impact	Contract fails to deliver promised returns, but doesn't lose value

Description

`FirelightVault` rejects the zero case on the value the caller receives in `deposit` and `mint`, which revert `InvalidAmount()` when the computed shares floor to zero ([FirelightVault.sol#L646-L700](#)). `redeem` guards only its input: it reverts on `shares == 0` but never checks the `assets` it computes by flooring ([FirelightVault.sol#L717-L744](#)), and `_requestWithdraw` then adds zero to every withdrawal bucket while still burning the shares through `_update(owner, address(0), shares)` ([FirelightVault.sol#L994-L1028](#)). When `totalAssets` sits below `totalSupply` — the expected state after a `payout` loss — any share amount that floors to zero assets is destroyed with nothing queued and nothing later claimable. The PoC redeems one share at a 0.01 share price: `totalSupply` falls from 100 to 99 while `withdrawSharesOf` and `withdrawAssets` stay zero, and a third party spending an allowance can do the same to a victim's shares. The value accrues to remaining holders, so the contract fails to deliver promised returns, but doesn't lose value.

IMM-SC-INSIGHT-88930

NatSpec across `FirelightVault`, `ICoverOrderAllocator` and `IIncidentManager` describes behaviour the implementations do not have

Report ID	#88930
Severity	Insight
Category	Blockchain/DLT
Asset	https://github.com/immuneFi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/interfaces/IIncidentManager.sol
Impact	—

Description

Six public NatSpec comments describe behaviour their implementations do not have:

`getEffectiveCapacityConfig` is documented as the current capacity configuration but returns `_getEffectiveCapacityConfig(_currentPeriod() + 1)` ([ICoverOrderAllocator.sol#L713-L714](#), [CoverOrderAllocator.sol#L940-L942](#)), and `deposit` documents only blacklist and pause checks while also reverting `CurrentPeriodHasActiveIncident()` ([FirelightVault.sol#L641-L660](#)). Integrators, auditors and off-chain tooling that reason from the published interface rather than the implementation are misled about actual contract behaviour.

IMM-SC-INSIGHT-88963

`CapacityConfig` field order leaves `divergenceToleranceBps` in a storage slot of its own

Report ID	#88963
Severity	Insight
Category	Blockchain/DLT
Asset	https://github.com/immunefi-team/audit-comp-firelight/blob/v1_audit_ready/contracts/core/interfaces/ICoverOrderAllocator.sol
Impact	Unbounded gas consumption

Description

`CapacityConfig` declares `uint16 divergenceToleranceBps` as its last field, after `firstLossBufferToken` and three `uint256`s, so it opens a sixth storage slot instead of packing into the 12 free bytes beside that address ([ICoverOrderAllocator.sol#L176-L191](#)). A two-harness benchmark measures 21,790 extra gas per `capacityConfigHistory.push` ([CoverOrderAllocator.sol#L673-L674](#)) and 2,115 per cold read, the unbounded gas consumption falling on every allocation commit and assessment approval.