F Y E O

# FLARE

Repo: https://github.com/mboben/avalanchego

Security Review Update: September 29, 2025

Reviewer: balthasar@gofyeo.com

# Security Code Review FLARE

## New security issues, 6

After the development team implemented the latest updates, FYEO conducted a review of the modifications. The primary goal of this evaluation was to ensure the continued robustness of the program's security features, safeguarding user funds and maintaining the overall integrity of the program.

### General Updates:

The changes perform a broad, cross-cutting refactor of how validator weights and related counts are represented and handled across the codebase: many components (APIs, trackers, sync/pathing, throttlers, metrics and database helpers) move from fixed-width integers to arbitrary-precision integers. That work standardizes weight arithmetic and quorum/startup calculations, and ripples into API responses and allocation logic, so numeric handling and some public representations have changed.

Alongside that refactor the patch expands multi-network support and behavior: new genesis/config entries and network targets were added (Flare, Songbird, Costwo/Coston and local variants), and platform VM logic gained per-network, time-aware inflation and staking rules that the verifier now consults. A few compatibility and UX tweaks were included as well (more complete validator replies behind a flag, a small secp256k1 signing fallback for broader signer compatibility, a network-aware application prefix helper, and some tightened config validations).

## CALLING DAEMON/MINT DURING STATE TRANSITION

Finding ID: FYEO-FLARE-01
Severity: **Low**
Status: **Acknowledged**

### Description

StateTransition.TransitionDb calls `atomicDaemonAndMint(st, log)` while processing a transaction (inside TransitionDb) — after successful VM call but before returning the execution result. The daemon/mint operation appears to change chain state (mint balances). Running minting while a transaction is being processed risks concurrent or out-of-order state changes that can break consensus or introduce race conditions (different execution order between nodes).

### Proof of Issue

**File name:** core/state_transition.go

```
// Call the daemon if there is no vm error
if vmerr == nil && (isSongbird || isFlare) {
    log := log.Root()
    atomicDaemonAndMint(st, log)
}
```

### Severity and Impact Summary

Potential consensus divergence or non-deterministic block processing. If daemon/mint mutates global balances/state that other parts of the transaction processing expect to be stable, nodes may apply different state changes in different orders (or at different times), causing forked state or invalid blocks.

### Recommendation

Do not perform non-deterministic or external daemon state changes inside per-transaction `TransitionDb` flow. Move daemon/mint calls into a deterministic block finalisation path where order is well-defined and identical across validators.
Add tests that assert identical post-block state across independent runs and verify no concurrent access to shared state during TransitionDb.

## API COMPATIBILITY & JSON ENCODING

Finding ID: FYEO-FLARE-02
Severity: **Informational**
Status: **Remediated**

### Description

Public ACP API fields were changed from `json.Uint64` (numeric JSON) to `*big.Int`. `big.Int` marshals to JSON as a string (via `TextMarshaler`), and a `nil *big.Int` marshals to `null`. This is a breaking change for API consumers who expect numeric JSON fields (and not strings or nulls).

#### Proof of Issue

**File name:** api/info/service.go

```
type ACP struct {
    SupportWeight *big.Int json:"supportWeight"
    ...
}
```

Old code used `json.Uint64` which encoded as JSON number.

### Severity and Impact Summary

API clients expecting numeric JSON will break (parsing errors or type mismatches). `nil` fields produce `null` — unexpected for numeric semantics.

### Recommendation

Either keep the public API numeric or implement custom `MarshalJSON` on ACP to emit a JSON number (or `0`) consistently. Ensure `nil` big.Ints are marshalled as `0` (not `null`) if that matches previous behavior.

## GetBigInt / ParseBigInt should treat empty DB value as zero

Finding ID: FYEO-FLARE-03
Severity: **Informational**
Status: **Remediated**

### Description

`GetBigInt` / `ParseBigInt` call `SetBytes(b)` directly. If the DB returns an empty `[]byte{}` (zero-length) the result may be `0` or ambiguous; currently code returns `new(big.Int).SetBytes(b)` without guarding for `len(b)==0`. Prefer to explicitly return `big.NewInt(0)` for empty input to avoid `nil` or undefined behaviour.

### Proof of Issue

**File name:** database/helpers.go

```
func GetBigInt(db KeyValueReader, key []byte) (*big.Int, error) {
    b, err := db.Get(key)
    if err != nil {
        return nil, err
    }
    return new(big.Int).SetBytes(b), nil
}
```

And ParseBigInt:

```
func ParseBigInt(b []byte) (*big.Int, error) {
    return new(big.Int).SetBytes(b), nil
}
```

No `len(b) == 0` guard.

### Severity and Impact Summary

Unexpected `nil` or zero handling may propagate and interact badly with consumers. Standardize on `0` for empty DB values to avoid panics or ambiguous semantics.

### Recommendation

Update both functions to treat empty byte slices as zero. Also consider returning `(*big.Int, bool, error)` where `bool` indicates presence vs missing key, if distinction between DB-missing and zero is important. Add unit tests for empty/absent keys.

## Inconsistent getTxFeeConfig condition vs genesis.GetTxFeeConfig

Finding ID: FYEO-FLARE-04
Severity: **Informational**
Status: **Remediated**

### Description

`getTxFeeConfig` was changed from returning a `TxFeeConfig` for many non-mainnet networks to returning it only when `networkID` equals `LocalFlareID` or `LocalID`. Meanwhile `genesis.GetTxFeeConfig` was modified to omit `LocalFlareID` and `LocalID` (comment: fees are set from program arguments).

This mismatch is confusing: either local networks should be handled by CLI args in which case both functions should omit them, or local networks should remain in the genesis mapping and `getTxFeeConfig` should not special case them.

### Proof of Issue

**File name:** config/config.go

```
-    if networkID != constants.MainnetID {
+    if networkID == constants.LocalFlareID || networkID == constants.LocalID {
        return genesis.TxFeeConfig{
            CreateAssetTxFee: v.GetUint64(CreateAssetTxFeeKey),
            StaticFeeConfig: txfee.StaticConfig{
                ...
            },
        }
    }
```

**File name:** genesis/params.go

```
func GetTxFeeConfig(networkID uint32) TxFeeConfig {
    // TxFee configs are ommitted for LocalFlare and Local networks since they
    // are set from program arguments (getTxFeeConfig)
    switch networkID {
    case constants.MainnetID:
        return MainnetParams.TxFeeConfig
    case constants.FlareID:
        return FlareParams.TxFeeConfig
    ...
    }
}
```

### Severity and Impact Summary

Networks other than `MainnetID`, `LocalFlareID`, and `LocalID` may not receive the intended `TxFeeConfig` from `getTxFeeConfig`.

### Recommendation

Choose one explicit, consistent approach and make the code reflect it.

## POSSIBLE NIL `*BIG.INT` DEREFERENCES

Finding ID: FYEO-FLARE-05
Severity: **Informational**
Status: **Remediated**

### Description

ACP numeric fields were changed from `json.Uint64` to `*big.Int`. Code performs in-place `Add`/`SetUint64` operations assuming the pointer is non-nil. If fields like `SupportWeight`, `ObjectWeight`, `AbstainWeight` are `nil`, calls such as `new(big.Int).Add(acp.SupportWeight, ...)` will panic.

### Proof of Issue

**File name:** api/info/service.go

```
type ACP struct {
    SupportWeight *big.Int json:"supportWeight"
    ...
}
...
acp.SupportWeight = new(big.Int).Add(acp.SupportWeight,
new(big.Int).SetUint64(weight))
```

Also `getACP` is expected to allocate defaults but code shows no explicit zero init.

### Severity and Impact Summary

Nil pointer dereference panics at runtime when building API responses — will crash the info service or handler and disrupt API availability.

### Recommendation

Ensure ACP instances initialize their big.Int fields to zero on creation.

## splitAllocations returns nil when numSplits is 0

Finding ID: FYEO-FLARE-06
Severity: **Informational**
Status: **Remediated**

### Description

`splitAllocations` currently returns `nil` when `numSplits == 0`. This silently returns nil instead of an error, which can cause nil deref or silent logical errors downstream.

### Proof of Issue

**File name:** genesis/genesis.go

```
func splitAllocations(allocations []Allocation, numSplits int) [][]Allocation {
    if numSplits == 0 {
        return nil
    }
    ...
}
```

### Severity and Impact Summary

Silent `nil` return can propagate and cause panics or unexpected behavior where caller expects a non-nil slice or an explicit error. It hides incorrect inputs.

### Recommendation

Validate input and return an explicit error or panic with a clear message.

## Commit Hash Reference:

For transparency and reference, the security review was conducted on the specific commit hashes for the following repositories. The commit hashes for the reviewed versions are as follows:

Avalancego:

https://github.com/ava-labs/avalanchego/compare/049be36ded7d35d0709f5f9b29917a3f9c5abed5...mboben:avalanchego:flare-merge-1_11_13

Coreeth:
https://github.com/ava-labs/coreth/compare/c06af9bb10be60babe9ecbdc2d0582cef68f40bc...mboben:coreth:flare-merge-0_13_9_rc1

Remediations were submitted with the commit hash f657f5c28720ec99d0c4e9c65190eb61a0011404.

## Conclusion:

In conclusion, the security aspects of the FLARE program remain robust and unaffected by the recent updates. Users can confidently interact with the protocol, assured that their funds are well-protected. The commitment to security exhibited by the development team is commendable, and we appreciate the ongoing efforts to prioritize the safeguarding of user assets.