OpenZeppelin | security

Avalanche Warp Messaging Security Review

Ava Labs.

November 16, 2023

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-10-02 To 2023-10-27
Languages	Go

Total Issues	3 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (0 resolved)
Notes & Additional Information	2 (0 resolved)

Scope

We audited the ava-labs/subnet-evm repository at commit c354ad6.

In scope were the following contracts:

```
contracts
├── warp
│   ├── aggregator
│   │   ├── aggregation_job.go
│   │   ├── aggregator.go
│   │   ├── network_signature_backend.go
│   │   └── signature_job.go
│   ├── handlers
│   │   ├── stats/stats.go
│   │   └── signature_request.go
│   ├── payload
│   │   ├── addressed_payload.go
│   │   ├── block_hash_payload.go
│   │   └── codec.go
│   ├── validators
│   │   └── state.go
│   ├── backend.go
│   ├── warp_client_fetcher.go
│   ├── warp_client.go
│   └── warp_service.go
└── x/warp
    ├── config.go
    ├── contract.abi
    ├── contract.go
    ├── contract_warp_handler.go
    └── module.go
```

System Overview

The Avalanche Warp Messaging is a foundational layer for cross-subnet communication within the Avalanche Network. The Warp protocol aims to establish secure and verifiable message passing between different Avalanche Subnets.

Warp leverages BLS multi-signatures and weighted validator sets to provide an efficient and secure communication layer. Every Subnet's validator set is indexed on the Avalanche P-Chain. When a validator joins a Subnet, its BLS public key is recorded which enables the P-Chain to keep track of the validator set.

Messages in Warp use a specific serialization format, which includes fields like `networkID`, `sourceChainID`, and an arbitrary payload. Messages are first created in the source chain by interacting with the Warp precompile, emitting the unsigned Warp message as an event. It is then the job of relayers to pick these unsigned messages and get enough individual validators to sign them until the destination chain's required stake threshold is reached. Once sufficient signatures are acquired, the relayer will aggregate the BLS signatures in a single, aggregated BLS signature to produce a signed message, alongside a bitset indicating which validators' signatures are included.

The signed message, as well as the bitset of validators, is encoded as a predicate and included in the Access List (EIP-2930) of a transaction, which is submitted to the receiving chain for verification and processing. If the predicate encoding and BLS aggregate signature are valid, the receiving chain will be able to access the message through the precompile interface and process it accordingly.

Security Model and Trust Assumptions

Lack of Built-in Replay Attack Prevention

The Warp protocol does not inherently provide mechanisms to prevent replay attacks. In a replay attack, a malicious actor could resend a previously sent valid message, potentially

causing unintended side effects or state changes on the receiving blockchain. The responsibility of preventing replay attacks is left to the implementation. As such, it is imperative for developers and system architects to be aware of this limitation and actively incorporate additional measures to mitigate the risk of replay attacks.

Lack of Destination Chain Validation

Warp does not include built-in checks or constraints to validate the destination chain for which a message is intended. This means that a message originally intended to be sent from Chain A to Chain B could potentially be received and processed by Chain C or any other chain within the Avalanche Network. This responsibility is also left to the implementation. The lack of this feature poses a security risk where a message could be wrongly interpreted or executed by an unintended recipient chain. This can lead to unintended state changes or even asset misallocations.

Disclaimer: *Please keep in mind that a security review is not a substitute for an audit, where the codebase is explored in a deeper technical level and with more time. While OpenZeppelin made its best efforts to share any situations that may constitute issues or areas of improvement, we cannot guarantee that all such issues have been detected.*

Low Severity

L-01 Potential Denial of Service Due to Limited Cache for Validator Signatures

Relayers request validators to sign valid warp messages. Afterwards, they aggregate the BLS signatures from all signers, and submit a transaction to the destination chain with the aggregated signatures encoded in the `AccessList` as a predicate.

Validators store unsigned warp messages in persistent storage. In addition, two limited-size caches are used to store signatures and unsigned warp messages, which are both cleared upon a system restart. Signatures are only cached due to the fact that the validator BLS keys may change during a restart, which would result in a different signature for the same message and thus render the signatures in the cache invalid.

Given that generating signed messages is computationally expensive for the validators, it is possible for a malicious relayer to exploit the cache limitation. This can be done by requesting signatures for a large number of warp messages, thereby exceeding the cache size. The malicious validator can keep cycling the large number so that validators keep having to compute the signature for a message and can't simply return the response from the cache.

This can consume the network's resources and potentially degrade the performance or availability of the network.

Consider setting up network monitoring and expanding the current DoS protection measures to prevent this scenario.

Update: *Acknowledged, not resolved.*

Notes & Additional Information

N-01 Inefficient Struct Memory Layouts

Several structs within the codebase are not optimally organized, leading to wasted memory due to padding and alignment. This results in higher-than-necessary memory usage, which can affect performance and resource utilization:

- `signatureAggregationJob`: 104 pointer bytes, could be 40
- `AggregateSignatureResult`: 24 pointer bytes, could be 8
- `Aggregator`: 64 pointer bytes, could be 32
- `signatureJob`: 56 pointer bytes, could be 32
- `AddressedPayload`: 104 pointer bytes, could be 32
- `BlockHashPayload`: 40 pointer bytes, could be 8
- `WarpMessage`: 18: 112 pointer bytes, could be 8
- `SendWarpMessageInput`: 64 pointer bytes, could be 8

Consider reordering `struct` fields to place the largest data types at the beginning and group fields of similar sizes together. The alignment can be verified with the `fieldalignment` tool. This will help minimize padding and optimize memory usage.

Update: *Acknowledged, not resolved.*

N-02 Unnecessary Assignment of Empty Signature

When a relayer asks a node for the signature of a specific unsigned warp message, the `OnSignatureRequest` function is called. This function will retrieve a BLS signature for a requested message ID. The function will ask the backend for a signature by calling the `GetSignature` function. In case this function returns an error, the signature is set to an

empty byte array of 96 bytes. However, this assignment is unnecessary as the `GetSignature` function of the backend already sets the signature to an empty byte array of 96 bytes in case the unsigned warp message is not found or the backend fails to sign it.

Consider removing the assignment of the empty byte array in the `OnSignatureRequest` function of the `signatureRequestHandler` struct.

Update: *Acknowledged, not resolved.*

Conclusion

The Avalanche Warp Messaging protocol provides a minimalistic yet robust design which serves as a base for future projects to build cross-chain messaging on top of. Due to the fact that the protocol implementation only contains the essentials, it is crucial for developers to consider security aspects while building upon it. Ava Labs has thoroughly documented these security concerns and has also provided a sample implementation called Teleporter that addresses all those considerations.

During the audit, it was discovered that a comprehensive review required exploring the codebase beyond the original scope. Nevertheless, the Ava Labs team was remarkably responsive and effective in their communication. A stand-out aspect of the protocol was the high quality of the codebase. The code is not only well-written, but also demonstrates a nuanced understanding of potential edge cases which have been both identified and extensively documented.