MONETHIC

Arkeo

Blockchain Audit Report

Prepared for:
Arkeo

Date:
28.02.2025

Version:
Final, for public release

Table of Contents

About Monethic

Monethic is a young and thriving cybersecurity company with extensive experience in various fields, including Smart Contracts, Blockchain protocols (layer 0/1/2), wallets and off-chain components audits, as well as traditional security research, starting from penetration testing services, ending at Red Team campaigns. Our team of cybersecurity experts includes experienced blockchain auditors, penetration testers, and security researchers with a deep understanding of the security risks and challenges in the rapidly evolving IT landscape. We work with a wide range of clients, including fintechs, blockchain startups, decentralized finance (DeFi) platforms, and established enterprises, to provide comprehensive security assessments that help mitigate the risks of cyberattacks, data breaches, and financial losses.

At **Monethic**, we take a collaborative approach to security assessments, working closely with our clients to understand their specific needs and tailor our assessments accordingly. Our goal is to provide actionable recommendations and insights that help our clients make informed decisions about their security posture, while minimizing the risk of security incidents and financial losses.

About Project

Arkeo is a free-market, decentralized network that provides access to blockchain data. Its intended objective is to provide a decentralized option for communicating with any blockchain (including but not limited to) and its data. It is comparable to Infura and Alchemy, which are currently the industry's leading providers of blockchain data.

Disclaimer

This report reflects a rigorous security assessment conducted on the specified product, utilizing industry-leading methodologies. While the service was carried out with the utmost care and proficiency, it is essential to recognize that no security verification can guarantee 100% immunity from vulnerabilities or risks.

Security is a dynamic and ever-evolving field. Even with substantial expertise, it is impossible to predict or uncover all future vulnerabilities. Regular and varied security assessments should be performed throughout the code development lifecycle, and engaging different auditors is advisable to obtain a more robust security posture.

This assessment is limited to the defined scope and does not encompass parts of the system or third-party components not explicitly included. It does not provide legal assurance of compliance with regulations or standards, and the client remains responsible for implementing recommendations and continuous security practices.

Scoping Details

The purpose of the assessment was to conduct a second Blockchain Security Audit against Arkeo chain, shared with Monethic through the GitHub platform.

Scope

The scope of the assessment includes the repository listed below:

- https://github.com/arkeonetwork/arkeo
    - `7338e8582a6f58bc0c82c902c4af56b3ecf8431c`

Scope of the assessment is agreed as whole repository security audit, extended by:

- Running `arkeod` in testnet environment,
- Any exploits around the code or things that can be accomplished on testnet,
- Testing opening/closing contracts (and seeing if any contracts can be exploited in unexpected ways)

Official Arkeo technical documentation was provided:

- https://docs.arkeo.network/

Timeframe

On January 15th Monethic was requested for the second security audit of Arkeo. Work began January 24th.

On February 11th, the Draft report from the Blockchain security assessment was delivered to the Customer.

Between February 21st and February 28th, retests of the fixes provided by the Customer were performed.

On February 28th, the Final report from the Blockchain security assessment was delivered to the Customer.

Vulnerability Classification

All vulnerabilities described in the report were thoroughly classified in terms of the risk they generate in relation to the security of the contract implementation. Depending on where they occur, their rating can be estimated on the basis of different methodologies. In most cases, the estimation is done by summarizing the impact of the vulnerability and its likelihood of occurrence. The table below presents a simplified risk determination model for individual calculations.

		Impact		
	Severity	**High**	**Medium**	**Low**
Likelihood	**High**	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Vulnerabilities that do not have a direct security impact, but may affect overall code quality, as well as open doors for other potential vulnerabilities, are classified as **Informational**.

Vulnerabilities summary

No.	Severity	Name	Status
1	Critical	Performing the same address transfer in `MsgClaimThorchain` results in free rewards	**Resolved**
2	High	Potential chain halt due to non-determinism and local time usage	**Resolved**
3	High	Incorrect reward distribution logic in `ValidatorPayout`	**Resolved**
4	High	Signatures can be replayed across different networks	**Resolved**
5	Medium	Inconsistent implementation compared with documentation	**Resolved**
6	Medium	Fees are stuck in the `arkeo` module	**Resolved**
7	Medium	`VersionForAddress` is not included when importing and exporting the genesis state	**Resolved**
8	Medium	Vulnerabilities in outdated dependencies	**Resolved**
9	Medium	`HasBeginBlocker` extension interface is not implemented in `x/arkeo`	**Resolved**
10	Medium	Potential incorrect `AmountClaim.IsZero()` condition in the `x/claim` module	**Resolved**
11	Low	The recipient's `IsTransferable` field is overwritten to false in `MsgClaimThorchain`	**Resolved**
12	Low	Delegate feature must be used during `MsgOpenContract`	**Resolved**
13	Low	Old expirations are not removed	**Resolved**
14	Low	Missing query commands implementation in `arkeo` module	**Resolved**

15	Low	Missing validation in `ValidateGenesis`	**Resolved**
16	Informational	Inconsistent `ServiceLookup` and `ServiceReverseLookup` map	**Resolved**
17	Informational	`ErrAirdropEnded` error is registered as 1	**Resolved**
18	Informational	Code quality improvements	**Resolved**

Technical summary

1. Performing the same address transfer in `MsgClaimThorchain` results in free rewards

Severity: Critical

Location

- `x/claim/keeper/msg_server_claim_thorchain.go`

Description

The `ClaimThorchain` function allows the Thorchain server to transfer funds from `msg.FromAddress` to `msg.ToAddress`. According to the documentation, this mechanism is added to allow THORChain users to transfer their airdrop from the initially allocated address derived using the THORChain coin type to their preferred Arkeo address.

The issue is that if the `msg.FromAddress` to the `msg.ToAddress` fields are specified as the same address, the `ClaimThorchain` function will incorrectly grant free coins to the recipient. This is because the `SetClaimRecord` function will overwrite the recipient's claim record with the accumulated rewards amount, incorrectly double-counting the user's rewards.

Test Case

```
func TestClaimThorchainSameAddress(t *testing.T) {
	msgServer, keepers, ctx := setupMsgServer(t)
	sdkCtx := sdk.UnwrapSDKContext(ctx)

	config := sdk.GetConfig()
	config.SetBech32PrefixForAccount("arkeo", "arkeopub")

	arkeoServerAddress, err :=
sdk.AccAddressFromBech32("arkeo1z02ke8639m47g9dfrheegr2u9zecegt50fjg7v")
	require.NoError(t, err)

	fromAddr := utils.GetRandomArkeoAddress()
	// toAddr := utils.GetRandomArkeoAddress()
	claimRecordFrom := types.ClaimRecord{
		Chain:          types.ARKEO,
		Address:        fromAddr.String(),
		AmountClaim:    sdk.NewInt64Coin(types.DefaultClaimDenom, 100),
		AmountVote:     sdk.NewInt64Coin(types.DefaultClaimDenom, 100),
		AmountDelegate: sdk.NewInt64Coin(types.DefaultClaimDenom, 100),
	}

	err = keepers.ClaimKeeper.SetClaimRecord(sdkCtx, claimRecordFrom)
	require.NoError(t, err)

	// mint coins to module account
	err = keepers.BankKeeper.MintCoins(sdkCtx, types.ModuleName,
sdk.NewCoins(sdk.NewInt64Coin(types.DefaultClaimDenom, 10000)))
	require.NoError(t, err)

	claimMessage := types.MsgClaimThorchain{
		Creator:     arkeoServerAddress.String(),
		FromAddress: fromAddr.String(),
		ToAddress:   fromAddr.String(), // same address
	}
	response, err := msgServer.ClaimThorchain(ctx, &claimMessage)
	require.NoError(t, err)
	require.NotNil(t, response)
	require.Equal(t, fromAddr.String(), response.FromAddress)

	// check if claimrecord is updated
	claimRecordFrom, err = keepers.ClaimKeeper.GetClaimRecord(sdkCtx,
fromAddr.String(), types.ARKEO)
	require.NoError(t, err)
	require.True(t, !claimRecordFrom.IsEmpty())
	require.Equal(t, claimRecordFrom.Address, fromAddr.String())
	require.Equal(t, claimRecordFrom.Chain, types.ARKEO)
	require.Equal(t, claimRecordFrom.AmountClaim,
sdk.NewInt64Coin(types.DefaultClaimDenom, 200))
	require.Equal(t, claimRecordFrom.AmountVote,
sdk.NewInt64Coin(types.DefaultClaimDenom, 200))
	require.Equal(t, claimRecordFrom.AmountDelegate,
sdk.NewInt64Coin(types.DefaultClaimDenom, 200))

}
```

Remediation

We recommend returning an error if the `msg.FromAddress` and `msg.ToAddress` addresses are identical.

Status

Resolved in 05bcfe89d9e779e62d2e0fa25051a68d2c56ddee.

2. Potential chain halt due to non-determinism and local time usage

Severity: High

Location

- `x/arkeo/configs/config.go:76-90`
- `x/claim/keeper/claim.go:70-80`
- `x/claim/keeper/claim.go:112-121`
- `x/claim/types/params.go:27`

Description

The codebase contains functions that use non-determinism and variables that attempt to access local time. This is problematic because if these functions or variables are used in production, a consensus failure will trigger, causing a chain halt.

Firstly, the `String()` function in `x/arkeo/configs/config.go:76-90` iterates over the `cv.int64values` and `cv.boolValues` maps when printing the `ConfigVals` values. This will cause the `sb.String()` output to be different across validators, causing an app-hash mismatch and a chain halt.

Secondly, the `GetAllClaimRecords` function in `x/claim/keeper/claim.go:70-80` iterates over the `types.Chain_name` mapping. This will cause the `claimRecords` output to be different across validators, causing an app-hash mismatch and a chain halt.

Thirdly, the `GetUserTotalClaimable` function in `x/claim/keeper/claim.go:112-121` iterates over the `types.Chain_name` mapping. If an error is returned in line `114` for any of the chain types, some validators will experience the error first, while others will not. This causes an app-hash mismatch and a chain halt.

Lastly, the `DeafultAirdropStartTime` variable in `x/claim/types/params.go:27` attempts to access the local time of validators via `time.Now().UTC()`. This value will be different across validators as they are spread across the globe with different timezones, causing an app-hash mismatch and a chain halt.

Remediation

We recommend sorting the mappings before performing the iteration and modifying the `DeafultAirdropStartTime` variable to use a hardcoded timestamp.

Status

Resolved in c680b2bdfe896bf6467867fa0962218b87777a8c.

3. Incorrect reward distribution logic in `ValidatorPayout`

Severity: High

Location

- `x/arkeo/keeper/manager.go`

Description

The `ValidatorPayout` function attempts to distribute blockReward to individual delegators and the validator itself by iterating over all delegations and taking into account the validator's commission amount. After computing the amount to transfer for each account, the funds are transferred via the `AllocateTokensToValidator` function.

The issue is that the `AllocateTokensToValidator` function does not send the rewards to individual accounts as intended. Instead, it distributes the rewards to the validator and the rest of the delegators after subtracting the validator's commission, which is the same logic that the `ValidatorPayout` function wants to implement.

This means that the `ValidatorPayout` logic will not work as intended, causing incorrect reward amounts to be transferred.

Remediation

We recommend sending the `totalReward` amount to the `x/distribution` module and call the `AllocateTokensToValidator` function directly.

Status

Resolved in 7869f4fee12e32397ba80f9b56dd22e074886bc2.

4. Signatures can be replayed across different networks

Severity: High

Location

- `x/arkeo/keeper/msg_server_claim_contract_income.go:59`

Description

The `HandlerClaimContractIncome` function verifies the signature for the `ContractAuthorization_STRICT` subscriptions to ensure that only authenticated users can claim contract incomes.

The issue is that the message to sign does not include the `chain ID` field. In `x/arkeo/types/message_claim_contract_income.go:52`, the message to sign only consists of the `contract ID` and the `nonce` field. This means that users could replay signatures from testnet to mainnet to bypass the signature verification mechanism.

Remediation

We recommend including the `chain ID` field in the required message to sign and optionally implement an expiration timeline.

Status

Resolved in 0257aa771230a6065b41bfba719f8feda161850d.

5. Inconsistent implementation compared with documentation

Severity: Medium

Description

There are a few sections where the documentation does not match the code implementation.

Firstly, the documentation mentions that data providers can cancel contracts at any time, as mentioned in:

"*Providers can cancel these contracts at any time*" and "*Pay-as-you-go isn't available to cancel as you can stop making requests as a form of canceling (providers can cancel though)*".

However, this is incorrect because only the client can cancel the contracts, as seen in `x/arkeo/keeper/msg_server_close_contract.go:59`.

Secondly, the `BlockPerYear` configured is `6311520` in `x/arkeo/types/params.go:18` and `x/arkeo/configs/config_v1.go:20`. However, the documentation states that it should be `5,256,666`, as seen in:

https://docs.arkeo.network/architecture/token#emission-formula.

Lastly, the documentation mentions that "*After 6 months from launch, all unclaimed airdrop tokens are sent back to the reserve*". However, no functionality in the codebase has been implemented to handle this. If the airdrop has ended, unclaimed funds remain in the `claimarkeo` module account. Ideally, unclaimed funds should be transferred to the `arkeo-reserve` module.

Remediation

We recommend updating the documentation to be consistent with the code behavior or modifying the implementation to be consistent with the documentation.

Status

Resolved in 6f3f18d17b1134e830a10436e97c77c21deb2992.

6. Fees are stuck in the `arkeo` module

Severity: Medium

Location

- `x/arkeo/keeper/manager.go`
- `x/arkeo/keeper/msg_server_open_contract.go`

Description

In the `SettleContract` and `OpenContractHandle` functions, the fees are sent to the `arkeo` module. However, these fees are not transferred to other modules and will be stuck there.

Remediation

We recommend sending the fees to the `arkeo-reserve` module.

Status

Resolved in 460dcfdb39d3323bf549182ee3a97358e496dba6.

7. `VersionForAddress` is not included when importing and exporting the genesis state

Severity: **Medium**

Location

- `x/arkeo/genesis.go`

Description

The `InitGenesis` and `ExportGenesis` functions do not support importing and exporting genesis for `VersionForAddress` (see `x/arkeo/keeper/keeper.go:257`). This means that the version chosen by the validator address via `MsgSetVersion` will not persist after a chain restart, harming the integrity of the genesis state.

Remediation

We recommend adding a field to the `GenesisState` struct to support importing and exporting the `VersionForAddress` state for validators.

Status

Resolved in bd35978525ab49cd4dc99fb88b06f5ab2a02a2fd.

8. Vulnerabilities in outdated dependencies

Severity: Medium

Location

- `go.mod`

Description

There are multiple outdated dependencies found in the codebase, opening up risks to vulnerabilities:

- ASA-2024-0012, ASA-2024-0013: Transaction decoding may result in a stack overflow or resource exhaustion
  - cosmossdk.io/x/tx@v0.13.3
  - github.com/cosmos/cosmos-sdk@v0.50.8
- ASA-2024-005: Potential slashing evasion during re-delegation
  - github.com/cosmos/cosmos-sdk@v0.50.8
- ASA-2024-010: cosmossdk.io/math: Mismatched bit-length validation in sdk.Int and sdk.Dec can lead to panic
  - cosmossdk.io/math@v1.3.0
- ASA-2024-009: CometBFT's state syncing validator from malicious node may lead to a chain split
  - github.com/cometbft/cometbft@v0.38.10
- CVE-2023-40591: Go-Ethereum vulnerable to denial of service via malicious p2p message
  - github.com/ethereum/go-ethereum@v1.10.17
- CVE-2024-32972: go-ethereum vulnerable to DoS via malicious p2p message
  - github.com/ethereum/go-ethereum@v1.10.17

Remediation

We recommend updating the dependencies to their latest versions:

- Update `cosmossdk.io/x/tx` to v0.13.7
- Update `github.com/cosmos/cosmos-sdk` to v0.50.11
- Update `cosmossdk.io/math` to v1.4.0
- Update `github.com/cometbft/cometbft` to v0.38.12
- Update `github.com/ethereum/go-ethereum` to v1.13.15

Status

Resolved in 77aaca10ad170e8e3ffa71bb38cc6d9a016c8848.

9. `HasBeginBlocker` extension interface is not implemented in `x/arkeo`

Severity: **Medium**

Location

- `x/arkeo/module.go`

Description

The `x/arkeo` module implements the `BeginBlock` function. However, the HasBeginBlocker extension interface is not defined in the module.

As a result, the chain will not call the `BeginBlock` function of the `x/arkeo` module, potentially causing unintended bugs, as seen here and here.

Remediation

We recommend defining the `HasBeginBlocker` extension interface.

Status

Resolved in 7869f4fee12e32397ba80f9b56dd22e074886bc2.

10. Potential incorrect `AmountClaim.IsZero()` condition in the `x/claim` module

Severity: Medium

Location

- `x/claim/keeper/msg_server_claim_eth.go:27`
- `x/claim/keeper/msg_server_claim_thorchain.go:26`
- `x/claim/keeper/msg_server_transfer_claim.go:21`

Description

The `MsgClaimEth`, `MsgClaimThorchain`, and `MsgTransferClaim` messages return an error if the `AmountClaim.IsZero()` condition is true. This is incorrect because the amount to claim or transfer may be `AmountVote` or `AmountDelegate` instead of `AmountClaim`.

Remediation

We recommend modifying the implementation to only return an error if `AmountClaim`, `AmountVote`, and `AmountDelegate` are zero.

Status

Resolved in 5354aa8230ff12e6f0631da2279f6ef7562dba4f.

11. The recipient's `IsTransferable` field is overwritten to false in `MsgClaimThorchain`

Severity: Low

Location

- `x/claim/keeper/msg_server_claim_thorchain.go:45`

Description

When combining the claim records, the `ClaimRecord.IsTransferable` field is not set to the `toAddressClaimRecord.IsTransferable` value. This is problematic because the recipient's `IsTransferable` field will be incorrectly overwritten to false, preventing them from calling `MsgTransferClaim`.

Remediation

We recommend setting the `ClaimRecord.IsTransferable` field to `toAddressClaimRecord.IsTransferable`.

Status

Resolved in c4d41cc17660e65aa33eaf0f21dc632689cfc94a.

12. Delegate feature must be used during `MsgOpenContract`

Severity: Low

Location

- `x/arkeo/keeper/msg_server_open_contract.go:111`

Description

The `OpenContractValidate` function calls the `msg.GetSpender()` function to determine the spender of the contract. If `msg.Delegate` is specified, the spender will be the delegatee instead of `msg.Client`. On the other hand, if the delegatee is not specified (i.e., the address is empty), the spender defaults to the client. Users can choose whether or not to use the delegate feature by setting the `msg.Delegate` field to be empty or a valid public key.

The issue is that line `155` of the `OpenContractHandle` function, which will be called after `OpenContractValidate`, casts the `msg.Delegate` field to a public key with the `NewPubKey` function. This effectively breaks the assumption that `msg.Delegate` can optionally be empty. Otherwise, the transaction fails.

Remediation

We recommend overwriting the `msg.Delegate` field to `msg.Client` if it is empty.

Status

Resolved in c4d41cc17660e65aa33eaf0f21dc632689cfc94a.

13. Old expirations are not removed

Severity: Low

Location

- `x/arkeo/keeper/manager.go:399`
- `x/arkeo/keeper/msg_server_close_contract.go:95`

Description

The `RemoveFromUserContractSet` function is called from the user's list of contracts if the contract is being settled. However, the `RemoveContractExpirationSet` function is not called to remove the contract's expiration height from the `prefixContractExpirationSet` state.

This will cause the `ContractEndBlock` function to iterate over settled contracts, unnecessarily increasing the computational power required.
Additionally, this issue also exists in the `CloseContractHandle` function, where the old expiration is not removed before setting a new expiration.

Remediation

We recommend calling the `RemoveContractExpirationSet` function to remove the contract's expiration height in the `ContractEndBlock` and `CloseContractHandle` functions.

Status

Resolved in c4d41cc17660e65aa33eaf0f21dc632689cfc94a.

14. Missing query commands implementation in `arkeo` module

Severity: Low

Location

- `x/arkeo/client/cli/query.go`

Description

The `arkeo` module implements a full range of client CLI commands that can be called, both state-changing and standard queries. While the state-changing commands are implemented correctly, four of the queries do not have a proper handler in the form of a `cmd.AddCommand` call, and as a result are not returned as available for use.

These are:

- `CmdShowContract`
- `CmdListContracts`
- `CmdShowProvider`
- `CmdListProviders`

Remediation

We recommend that the above commands be returned as possible to handle from the CLI level.

Status

Resolved in 614a1a18d756bb00540f4453913ff7719e39713b.

15. Missing validation in `ValidateGenesis`

Severity: Low

Location

- `x/arkeo/module.go`
- `x/claim/module.go`

Description

The `arkeo` and `claim` module implements `GenesisState` validation through the `ValidateGenesis` functions. They are calling the `genState.Validate`, which subsequently calls `gs.Params.Validate`.

This method, however, returns `nil` always, without any kind of validation on any of the `GenesisState` parameters. That might be problematic, as such validation should be in place to remove potentially bad genesis states.

Remediation

We recommend the implementation of basic validation to ensure that `GenesisState` is properly used during genesis.

Status

Resolved in c4d41cc17660e65aa33eaf0f21dc632689cfc94a.

16. Inconsistent `ServiceLookup` and `ServiceReverseLookup` map

Severity: Informational

Location

- `common/service.go`

Description

The `ServiceLookup` and `ServiceReverseLookup` map works interchangeably. However, there are inconsistencies in both mappings:

- The `ServiceLookup` mapping does not map the `bnb-mainnet-unchained` service to `35`, which is implemented in the `ServiceReverseLookup` mapping in line `108`.

```
var ServiceLookup = map[string]int32{
...
"btc-mainnet-unchained":        34,
"bsc-mainnet-unchained":        36,
...
}

var ServiceReverseLookup = map[Service]string{
...
35: "bnb-mainnet-unchained",
...
}
```

- The `ServiceReverseLookup` mapping contains duplicates of `bsc-mainnet-unchained` services, which should be removed.

```
var ServiceReverseLookup = map[Service]string{
...
36: "bsc-mainnet-unchained",
37: "bsc-mainnet-unchained",
...
}
```

Remediation

We recommend adding 35: `bnb-mainnet-unchained` to the `ServiceLookup` map and removing 37: "`bsc-mainnet-unchained`" from the `ServiceReverseLookup` maps.

Status

Resolved in 8a2b13b24acd9ad7c26e6606d19462355635ee91.

17. `ErrAirdropEnded` error is registered as `1`

Severity: **Informational**

Location

- `x/claim/types/errors.go:8`

Description

The `ErrAirdropEnded` error is registered as a code value of 1. This is problematic because error codes should be registered with a value greater than one, as a code value of one is reserved for internal errors.

```
ErrAirdropEnded  = errors.Register(ModuleName, 1, "Airdrop has ended")
```

Remediation

We recommend updating the `ErrAirdropEnded` error to use a value greater than 1.

Status

Resolved in 8a2b13b24acd9ad7c26e6606d19462355635ee91.

18. Code quality improvements

Severity: Informational

Description

The following instances represent code quality that can be improved:

- The validation in `x/arkeo/keeper/manager.go:249-252` can be removed because it was already validated previously in `x/arkeo/keeper/manager.go:227`.
- The validation in `x/arkeo/keeper/msg_server_close_contract.go:88` can be removed because it was already validated previously in `x/arkeo/keeper/msg_server_close_contract.go:58`.
- The validation in `x/claim/keeper/msg_server_claim_eth.go:146` can be removed because it was already validated previously in line `142`.
- The error message in `x/arkeo/keeper/msg_server_open_contract.go:95` should be modified to include “rate*duration*qpm” instead of “rate*duration” only.
- The error message in `x/arkeo/types/message_mod_provider.go:115` should be modified to “invalid pay-as-you-go rates”.
- Consider removing the `MsgAddClaim` logic in `x/claim/keeper/msg_server_add_claim_testnet.go:15`, as it should not be used in production.

Remediation

We recommend improving the code quality by applying the abovementioned suggestions.

Status

Resolved in 8a2b13b24acd9ad7c26e6606d19462355635ee91.

END OF THE REPORT