Security Audit Report

Symbiosis Finance Relayers v2

1. Contents

2. General Information

This report contains information about the results of the security audit of the Symbiosis Finance (hereafter referred to as "Customer") Relayers v2 and BTC Relayer, conducted by Decurity in the period from 06/14/2024 to 07/12/2024.

2.1. Introduction

Tasks solved during the work are:

- Review the application's business logic and potential risks,
- Perform penetration test against the API endpoints,
- Develop the recommendations and suggestions to improve the security of the API.

2.2. Scope of Work

The audit scope includes the following repositories:

- https://github.com/symbiosis-finance/relayer-public (commit 760e95630af7d8c568adfc8f33628f5e2eede3d6, 17 June 2024)
- https://github.com/symbiosis-finance/btc-relayer/pull/8 (commit 65617bbb6829f2f2363fb95557502783793d55de, 13 June 2024)

2.3. Threat Model

The assessment presumes actions of an external attacker who has no access to the application's source code or any privileged access to the infrastructure.

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered critical exploitable security issues which have been fixed and re-tested in the course of the work.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

The Symbiosis Finance team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues and their status as of August 17, 2024.

Table. Discovered weaknesses

Issue	Files	Status
[Relayer] An active staker can change the MPC address to the malicious one	relayer.v2/internal/epoch/change/change_sign.go	Fixed
[BTC-R] Integer overflow during wrap decoding	btc-relayer/internal/btcprovider/btcportalstate.go	Fixed
[Relayer] Epoch change DoS via event substitution in the txManager of a relayer	relayer.v2/internal/epoch/epoch_manager.go	Fixed
[Contracts] The stableBrigingFee is controlled by the user	contracts/synth-core/Synthesis.sol	Acknowledged
[Relayer] Broken P2P failedHandler access control	relayer.v2/internal/transaction/signing/transaction.go	Fixed

Issue	Files	Status
[BTC-R] DoS via FilterUnwraps	btc-relayer/internal/evmprovider/evmsynth.go	Fixed
[BTC-R] Panic in deposit handling	btc-relayer/internal/btcprovider/btcportalstate.go	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Follow the best practices of the secure software development (SDLC, BSIMM, SAMM, OWASP TG, etc),
- Keep the documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new features and updates,
- Launch a public bug bounty campaign for the applications and infrastructure.

5. Findings

5.1. [Relayer] An active staker can change the MPC address to the malicious one

Status: Fixed in the commit 64aa8e0e.

Files:

- relayer.v2/internal/epoch/change/change_sign.go

Location: Lines: `150`. Function: `sign()`.

Description:

Relayers have stream handlers `signEpochHandler` and `epochApprovedHandler` defined in the file `relayer.v2/internal/epoch/change/change_sign.go`. The first handler accepts a message with the new MPC address, MPC group, new epoch number, transaction hash and block number, where the event `PrepareEpochChange` was emitted. After some validations, the handler signs the message with current relayer's key and responses back with signed message to the peer called the handler.

It was discovered that validations performed in the handler (file `relayer.v2/internal/epoch/validator.go`) can be bypassed using specially crafted data:

1. An attacker may pass the new epoch number as the `current epoch + 1` without waiting for the corresponding event onchain. It will be accepted by the validator function `NewEpochNumber` and will help to bypass checks in functions `ActiveGroup` and `MPC`, since the epoch tx manager isn't created yet or doesn't contain the event information and new MPC address
2. The validator in the function `ActiveGroup` doesn't check that the passed transaction hash and block number with emitted event correspond to the new epoch generation process. An attacker may pass tx hash and block number from any old epoch generation, which will be accepted by the validator
3. Since there is no information about MPC address in the new epoch tx manager yet, the check in the `MPC` function will be skipped. As a result, any new value for the MPC address may be passed

4. The function `ActiveGroup` also collects active stakers from the Staking contract, sorts them based on the seed from the `PrepareEpochChange` event and checks that the passed `mpcGroup` is a subset of these stakers. Another requirement that should be satisfied is the number of veto staker entries in the group that should be equal or greater than `int(float64(vetoCount)/2) + 1`
   Summing up, the following attack can be executed:
5. The attacker doesn't wait for the next epoch change event onchain, creates a message with `NewEpochNumber == current epoch + 1`, `txHash` and `blockNumber` that contain an old `PrepareEpochChange` event, arbitrary malicious controllable MPC address, any MPC group that includes sufficient number of veto nodes and is a subset of the shuffling result based on the seed from the `PrepareEpochChange` event. The message is sent to all nodes in the MPC group on the handler `signEpochHandler`, then responses with signatures are collected
6. Having the signatures corresponding to the malicious MPC address and MPC group it is possible to change current epoch using the Staking contract function `changeEpoch`. After its execution, the MPC address will be changed to the malicious one and the `EpochChanged` event will be emitted
7. Current relayers network handles the emitted event and changes the MPC address to the malicious one on the Bridge contract in the same chain, which will allow the attacker to send arbitrary transactions to the bridge signed by the new controllable MPC address
8. Since the Bridge contract plays crucial role in the infrastructure and has high permissions over other contracts, it can be used to mint and burn synthetic tokens, withdraw funds from the same and other chains
   As a result, it is possible to steal all the funds from all chains by changing the MPC address.
   Remediation:
   - Check whether the current node has participated in the `MpcAddress` generation, and whether the other parameters (`NewEpochNumber`, `TxHash`, `BlockNumber`) match the corresponding event on chain.
   - Allow invoking `signEpochHandler` only to the leader.
   - Do not sign the epoch change message, if the node hasn't the MPC address in the new epoch tx manager set yet. Now lines 95-97 of the file

`relayer.v2/internal/epoch/validator.go` have the following code, which skips the check, if the MPC address isn't defined in the `txMgr`, since in this case `ok` will be equal to `false`:

```
} else if mpc, ok := txMgr.MpcAddr(); ok && mpc.Hex() != mpcAddr.Hex() {
    return fmt.Errorf("new mpc is different from the selfgenerated mpc, new
mpc: %s, self new mpc: %s", mpcAddr.Hex(), mpc.Hex())
}

Replace the code by the following one, which will throw an error if it is
impossible to obtain MPC address from the new epoch tx manager:

} else if mpc, ok := txMgr.MpcAddr(); !ok || mpc.Hex() != mpcAddr.Hex() {
    return fmt.Errorf("new mpc is different from the selfgenerated mpc, new
mpc: %s, self new mpc: %s", mpcAddr.Hex(), mpc.Hex())
}
```

5.2. [BTC-R] Integer overflow during wrap decoding

Status: Fixed in the commit e05276e2.

Files:

- btc-relayer/internal/btcprovider/btcportalstate.go

Location: Lines: 821.

Description:

Function `decodeWrap` in the `internal/btcprovider/btcportalstate.go` file does not account for overflows when computing the tx value after applying the fee:

```
Value:             types.Satoshi(tx.TxOut[idx].Value) - info.PortalFee,
```

The `Satoshi` type is integer, and the `info` structure is untrusted and controlled by the users, therefore, this value may be manipulated to negative.

As a result, during the EVM transaction serialization, this value will be casted to `uint256` and will become huge, leading to excessive minting of Synth BTC tokens.

Remediation:

Check for overflows when computing the amounts.

5.3. [Relayer] Epoch change DoS via event substitution in the txManager of a relayer

Status: Fixed in the commit 64aa8e0e.

Files:

- relayer.v2/internal/epoch/epoch_manager.go

Location: Lines: `130`. Function: `validateEpochChangeRequest()`.

Description:

Relayers have the `/p2p/relayer.v2/new_epoch_event/v1` stream handler that is set in the file `relayer.v2/internal/epoch/epoch_manager.go` . The handler accepts messages of the structure `MsgEpoch` that contains the `Event` field. This field is processed in the function `processEpochChangeRequest`, which checks the presence of the `txManager` for the `event.NextEpoch` epoch and creates it with corresponding `txHash` and `BlockNumber` of the `PrepareEpochChange` event transaction, if it doesn't exist. If it exists and the passed `txHash` is different than the stored one, the `txManager` will be recreated with new event data from the message.

The handler has the event validation performed in the `validateEpochChangeRequest` function, but it only checks the presence of the event on chain in `txHash` and `BlockNumber` specified by the caller. The `nextEpoch` argument in the event isn't checked and can be any old epoch number. Hence, an attacker can call the handler with old epoch change event data and create/replace the `txManager` for the specified epoch number.

This creates the DoS opportunity on the stage of signing the new `mpcGroup` and `mpcAddress` data for the next epoch, which involves the `/p2p/relayer.v2/sign_epoch_changing/v1` handler call, defined in the file `relayer.v2/internal/epoch/change/change_sign.go`. Since the handler takes the `seed` value from the event stored in the `txManager` and uses it to generate the active group, a wrong substituted event will result in a wrong seed and group generated. An attacker can wait for the new MPC signing stage, send the message to the `/p2p/relayer.v2/new_epoch_event/v1` handler on any relayer in the `mpcGroup` with any old `PrepareEpochChange` transaction hash and block number, create/recreate the `txManager` with wrong data, which will block the entire signing process of new MPC group and address.

Remediation:

Validate that the passed event with `txHash` and `BlockNumber` has the same next epoch number, as the event emitted on chain. This validation can be done in the function `validateEpochChangeRequest`

5.4. [Contracts] The stableBrigingFee is controlled by the user

Status: Acknowledged

Files:

- contracts/synth-core/Synthesis.sol

Location: Function: `mintSyntheticToken()`, `metaMintSyntheticToken()`, `metaMintSyntheticTokenBTC()`, `burnSyntheticToken()`, `metaBurnSyntheticToken()`.

Description:

The `stableBrigingFee` parameter is responsible for the fee that the user pays for executing a cross-chain operation. It was discovered that the parameter is set on the frontend while composing the transaction and is not checked anywhere except the `burnSyntheticTokenBTC` function of the `Synthesis` contract.

Since swaps are performed using the intermediate Boba BNB chain, where synthetic tokens are minted and burned, it is possible to mint synthetic tokens without fee for BTC operations (the fee isn't checked in the `metaMintSyntheticTokenBTC` function), as well as both mint and burn tokens for non-BTC operations without fee validation.

Such behavior allows using the protocol without fee. The fee is calculated before executing a cross-chain operation and includes evaluated gas cost that relayers should pay to perform the operation. Some chains, such as Ethereum, may have expensive gas price. Executing a lot of free operations that involve these chains will cause significant funds losses for the protocol

Remediation:

There are several approaches to fix the vulnerability. Some of them:

- Check the `stableBrigingFee` in the same way as it is implemented in the function `burnSyntheticTokenBTC` of the `Synthesis` contract using the separate syntToMinFeeBTC mapping. The disadvantage of such approach is the need to change the fee in the contract often according to current gas price on the corresponding chain

- Sign the fee, its expiration time, involved chain IDs and sender or destination address by a separate fee controller address on the backend and pass the signature to the cross-chain operation arguments. Validate the signature in the Portal and/or Synthesis contracts while executing the operation
- Estimate the gas cost before each transaction execution by relayers, and revert the entire operation, if the fee cannot cover the required execution cost

5.5. [Relayer] Broken P2P failedHandler access control

Status: Fixed in the commit 64aa8e0e. Now only leader can call this function.

Files:

- relayer.v2/internal/transaction/signing/transaction.go

Location: Lines: 645. Function: failedHandler.

Description:

The failedHandler p2p handler can be called by anyone in internal/transaction/signing/transaction.go:645:

```
func (t *Transaction) failedHandler(msg *struct{}, p peer.ID, logger
*zap.Logger) {
    t.failedBroadcast = true
    logger.Warn("command failed", zap.String("in_state",
t.fsm.CurrentState()))
    t.fsmFailed()
}
```

An attacker may call the handler during a transaction signing process without any permissions required, which will abort the process. Continuous monitoring for new transactions and calling the handler will cause DoS in the relayers network.

Remediation:

Limit access to the handler.

5.6. [BTC-R] DoS via FilterUnwraps

Status: Fixed in the commit 509fc65a.

Files:

- btc-relayer/internal/evmprovider/evmsynth.go

Description:

`FilterUnwraps` function in `btc-relayer/internal/evmprovider/evmsynth.go` returns nil wraps in case it fails to parse pkScript that is provided by user.

```
pkScript, err := txscript.ParsePkScript(event.To)
if err != nil {
    return nil, fmt.Errorf("failed to parse pkScript from BridgeV2 event %x:
%w", event.To, err)
}
```

This results in fact that if user provides invalid pkScript (e.g random data or unsupported type) unwrap processing will finish. Next wraps will not be processed, because evm unwrap anchor will not be updated in `finalizeUnwraps` due to the same error.

Remediation:

Consider just skipping invalid `pkScript` instead of returning nil wraps.

5.7. [BTC-R] Panic in deposit handling

Status: Fixed in the commit e05276e2.

Files:

- btc-relayer/internal/btcprovider/btcportalstate.go

Description:

User can craft a BTC transaction that will have 1 transaction in and 2 transactions out. In case both of transactions out go to portal address, `decodeDeposit` function at `btc-relayer/internal/btcprovider/btcportalstate.go` will throw panic. This will happen because it will try to access `tx.TxIn` at nonexistent index:

```
txIn := tx.TxIn[idx]
```

The index that is passed to decodeDeposit function is the index of an output transaction. However, in case when we have 1 transaction in with 2 outputs, length of `tx.TxIn` is 1. But, for the second output transaction provided index will be 1, which will result in an attempt to access array out of bounds.

Remediation:

Consider handling scenario with 2 output transactions to portal address.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.