Omniscia
SMART CONTRACT AUDITS

Smart Contract Security Assessment

18/03/2022

Prepared for
Symbiosis

Online report
symbiosis-finance-router-bridge

www. omniscia.io

info@omniscia.io

Meta Router Bridge Security Audit

Audit Overview

We were tasked with auditing the codebase of Symbiosis Finance and in particular the bridge and router modules meant to support their cross-chain synthetic asset system.

Over the course of the audit we identified a severe front-running vulnerability in the way reversions of relayed transactions occur that allow a user to cancel the transaction of another user arbitrarily.

Additionally, we were able to pinpoint several optimizations that can be applied across the codebase that we advise the Symbiosis Finance team to consider and apply along with remediations to all vulnerabilities identified within the report.

Post-Audit Conclusion

The Symbiosis Finance team remediated all the medium-severity and higher exhibits within the report adequately and alleviated a portion of the minor-to-informational severity findings according to their discretion.

The codebase can be considered of a high quality and adequately documented to be integrated by external projects.

The latest update to the codebase introduced graceful error handling that should not be considered as part of the audit scope.

Contracts Assessed

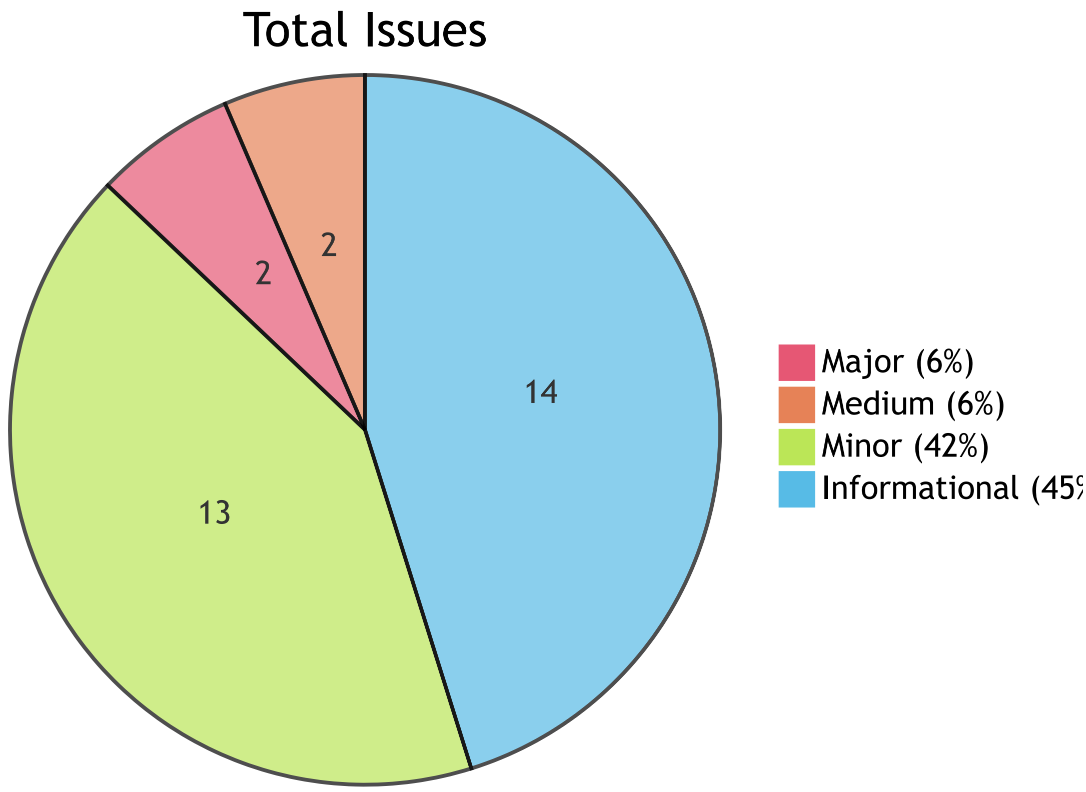
Files in Scope	Repository	Commit(s)
BridgeV2.sol (BV2)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
MetaRouterV2.sol (MRV)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
MetaRouteStructs.sol (MRS)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
Portal.sol (POR)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
RelayRecipientUpgradeable.sol (RRU)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
SyntERC20.sol (SER)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
Synthesis.sol (SYN)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
SyntFabric.sol (SFC)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2

Files in Scope	Repository	Commit(s)
Timelock.sol (TIM)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2
Wrapper.sol (WRA)	contracts-audit-with-tests	707f038827, 796b5eef15, dd00ff3939, 50dda9f9d2

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
● Major	2	2	0	0
● Medium	2	2	0	0
● Minor	13	7	0	0
● Informational	14	8	0	0

During the audit, we filtered and validated a total of **3 findings utilizing static analysis** tools as well as identified a total of **28 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they introduce potential misbehaviours of the system as well as exploits.



The list below covers each segment of the audit in depth and links to the respective chapter of the report:

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in TypeScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.0` for the subset of contracts within the audit scope based on the version specified within the `hardhat.config.ts` file.

The project contains discrepancies with regards to the Solidity version used as the contract's `pragma` statements are open-ended (`^0.8.0`).

We advise the `pragma` statements to be locked to `0.8.0` (`=0.8.0`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **309 potential issues** within the codebase of which **305 were ruled out to be false positives** or negligible findings.

The remaining **4 issues** were validated and grouped and formalized into the **3 exhibits** that follow:

ID	Severity	Addressed	Title
POR-01S	● Informational	✓ Yes	Leftover TODO Comment
POR-02S	● Informational	☒ No	Variable Shadowing
SYN-01S	● Informational	☒ No	Variable Shadowing

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in the cross-chain synthetic asset bridge.

As the project at hand implements a cross-chain aware bridge implementation, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification and that **all features exposed by it are blockchain-aware**.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **identified two vulnerabilities relating to access control** within the system which could have had **severe ramifications** to its overall operation, however, they were conveyed ahead of time to the Symbiosis Finance team to be **promptly remediated**.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a certain extent, however, we strongly recommend the documentation of the project to be expanded at certain complex points such as the function encoding for cross-chain interaction as those interfaces could not be validated by the codebase alone.

A total of **28 findings** were identified over the course of the manual review of which **17 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
BV2-01M	Medium	Yes	Inexistent Sanitization of Commissions
MRV-01M	Medium	Yes	Inexistent Validation of Calldata Slots
MRV-02M	Minor	No	Arbitrary Approvals
MRV-03M	Minor	No	Ill-Advised Allowance Pattern
MRV-04M	Minor	Yes	Improper `receive` Function
POR-01M	Major	Yes	Inexistent Access Control for Reverts
POR-02M	Minor	Yes	Improper `receive` Function
POR-03M	Minor	Yes	Potential of Repeat Invocation
SER-01M	Minor	No	Arbitrary Burn Operations
SYN-01M	Major	Yes	Inexistent Access Control for Reverts
SYN-02M	Minor	Yes	Improper Reversion of Burn
SYN-03M	Minor	Yes	Inconsistent Event Amount
SYN-04M	Minor	Yes	Inexistent Validation of Token Existence
SYN-05M	Minor	Yes	Potential of Repeat Invocation
WRA-01M	Minor	No	Deprecated Native Asset Transfer
WRA-02M	Minor	No	Improper `receive` Function
WRA-03M	Minor	No	Inexistent Validation of Amounts

Code Style

During the manual portion of the audit, we identified **11 optimizations** that can be applied to the codebase that will decrease the gas-cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
BV2-01C	Informational	No	Redundant Logical Block
MRV-01C	Informational	Yes	Data Location Optimization
MRV-02C	Informational	Yes	Redundant `constructor` Implementation
POR-01C	Informational	Yes	Inexistent Error Messages
RRU-01C	Informational	No	Redundant Implementation
RRU-02C	Informational	No	Redundant Import
SER-01C	Informational	Yes	Variable Mutability Specifier
SFC-01C	Informational	Yes	Inexistent Error Messages
SFC-02C	Informational	No	Inexistent Function Implementations
SYN-01C	Informational	Yes	Inexistent Error Messages
WRA-01C	Informational	Yes	Inexistent Visibility Specifier

Portal Static Analysis Findings

ON THIS PAGE

POR-01S: Leftover TODO Comment

Type	Severity	Location
Code Style	● Informational	Portal.sol:L301

Description:

The linked `TODO` comment indicates code that has not been clearly defined.

Example:

contracts/synth-contracts/Portal.sol

SOL

Copy

```sol
298 emit RevertSynthesizeCompleted(
299     _txID,
300     txState.recipient,
301     txState.amount, // TODO: which amount?
302     txState.rtoken
303 );
```

Recommendation:

We advise the proper `amount` event argument to be assessed, assimilated in the codebase and the comment to be removed.

Alleviation:

The amount argument was instead adjusted to one accounting for the stable bridging fee and the stable bridging fee is now minted along the `RevertSynthesizeCompleted` event.

POR-02S: Variable Shadowing

Type	Severity	Location
Language Specific	● Informational	Portal.sol:L100, L229

Description:

The linked variables cause a naming colission with equivalent-name variables in inherited implementations.

Example:

contracts/synth-contracts/Portal.sol

SOL

```sol
100 address _trustedForwarder,
```

Recommendation:

We advise them to be renamed to avoid the colission and potentially undefined code behaviour.

Alleviation:

The Symbiosis Finance team considered this exhibit but opted not to apply a remediation for it in the current iteration.

Synthesis Static Analysis Findings

ON THIS PAGE

SYN-01S: Variable Shadowing

SYN-01S: Variable Shadowing

Type	Severity	Location
Language Specific	● Informational	Synthesis.sol:L92

Description:

The linked variables cause a naming colission with equivalent-name variables in inherited implementations.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```sol
92 address _trustedForwarder,
```

Recommendation:

We advise them to be renamed to avoid the colission and potentially undefined code behaviour.

Alleviation:

The Symbiosis Finance team considered this exhibit but opted not to apply a remediation for it in the current iteration.

BridgeV2 Manual Review Findings

ON THIS PAGE

BV2-01M: Inexistent Sanitization of Commissions

Type	Severity	Location
Input Sanitization	● Medium	BridgeV2.sol:L97-L103, L172-L178

Description:

The linked functions allow either the MPC or the owner to request and receive their commissions, however, all input arguments are blindly trusted and no sanitization occurs on those values.

Example:

contracts/synth-contracts/bridge-v2/BridgeV2.sol

SOL

Copy

```
97  /**
98  * @notice Get commission by MPC
99   */
100 function getCommissionByMPC(address token, address to, uint256 amount) external onl
101     TransferHelper.safeTransfer(token, to, amount);
102     return true;
103 }
```

Recommendation:

As the contract is meant to retain funds at rest, we strongly advise this trait of the system to be re-evaluated and commissions to be tracked properly locally instead.

Alleviation:

The Symbiosis Finance team stated that this is intended behaviour as the contract is solely meant to retain commission funds at rest. As a result, we consider this exhibit null.

MetaRouterV2 Manual Review Findings

ON THIS PAGE

MRV-01M: Inexistent Validation of Calldata Slots

Type	Severity	Location
Input Sanitization	● Medium	MetaRouterV2.sol:L58, L83

Description:

The low level `assembly` writes to the two call datas are meant to fill in the value of a particular argument for the external call, however, no validation is performed on the calldata that can lead to out-of-bounds writes in the blocks as well as generally unexpected behaviour.

Example:

contracts/metarouter/MetaRouterV2.sol

SOL

Copy

```solidity
53 uint256 finalSwapAmountIn = secondSwapAmountIn;
54 if (_metarouteTransaction.secondSwapCalldata.length != 0) {
55     bytes memory secondSwapCalldata = _metarouteTransaction.secondSwapCalldata;
56
57     assembly {
58         mstore(add(secondSwapCalldata, 100), secondSwapAmountIn)
59     }
60
61     IERC20(_metarouteTransaction.approvedTokens[approvedTokensLength - 2]).approve(
62         _metarouteTransaction.secondDexRouter,
63         secondSwapAmountIn
64     );
65
66     (bool secondSwapSuccess,) = _metarouteTransaction.secondDexRouter.call(
67         secondSwapCalldata
68     );
```

```
    require(secondSwapSuccess, "MetaRouterV2: second swap failed");

    finalSwapAmountIn = IERC20(
        _metarouteTransaction.approvedTokens[approvedTokensLength - 1]
    ).balanceOf(address(this));
}

IERC20(_metarouteTransaction.approvedTokens[approvedTokensLength - 1]).approve(
    _metarouteTransaction.relayRecipient,
    finalSwapAmountIn
);

bytes memory otherSideCalldata = _metarouteTransaction.otherSideCalldata;
assembly {
    mstore(add(otherSideCalldata, 100), finalSwapAmountIn)
}
```

Recommendation:

We advise the calldata arguments to be validated by at least mandating they are of a particular `length`.

Alleviation:

After consideration of our exhibit & with the help of an external party, the Symbiosis Finance team identified a potential attack vector based on allowances set to the contract that arbitrary calls could exploit. The Symbiosis Finance team introduced the concept of a gateway contract that is meant to instead be set an allowance for by external users preventing the arbitrary calls performed by the `MetaRouterV2` contract to be able to tap into allowances set for it. Additionally, the two arbitrary calls performed now cannot have the gateway contract as a target thereby completely nullifying any attack vector that would affect user funds and rendering the contract secure. After additional discussion with the Symbiosis Finance team, we concluded that malicious data stacks for the linked `assembly` blocks would only affect the caller and would not pose a threat to other users or the network's state. As a result, this exhibit is considered dealt with.

MRV-02M: Arbitrary Approvals

Type	Severity	Location
Logical Fault	● Minor	MetaRouterV2.sol:L36-L39, L61-L64, L76-L79

Description:

The contract performs arbitrary `approve` invocations which allow crafted payloads to extract any funds at rest within the contract.

Example:

contracts/metarouter/MetaRouterV2.sol

SOL

Copy

```
18 function metaRouteV2(
19     MetaRouteStructs.MetaRouteTransactionV2 memory _metarouteTransaction
20 ) external payable {
21     uint256 firstSwapValue;
22     uint256 approvedTokensLength = _metarouteTransaction.approvedTokens.length;
23 
24     if (!_metarouteTransaction.nativeIn) {
25         TransferHelper.safeTransferFrom(
26             _metarouteTransaction.approvedTokens[0],
27             _msgSender(),
28             address(this),
29             _metarouteTransaction.amount
30         );
31     }
32 
33     uint256 secondSwapAmountIn = _metarouteTransaction.amount;
34     if (_metarouteTransaction.firstSwapCalldata.length != 0) {
35         if (!_metarouteTransaction.nativeIn) {
36             IERC20(_metarouteTransaction.approvedTokens[0]).approve(
37                 _metarouteTransaction.firstDexRouter,
38                 _metarouteTransaction.amount
39             );
40         }
41 
42         (bool firstSwapSuccess,) = _metarouteTransaction
43         .firstDexRouter
44         .call{value : msg.value}(_metarouteTransaction.firstSwapCalldata);
45 
46         require(firstSwapSuccess, "MetaRouterV2: first swap failed");
```

```

        secondSwapAmountIn = IERC20(
            _metarouteTransaction.approvedTokens[1]
        ).balanceOf(address(this));
    }

    uint256 finalSwapAmountIn = secondSwapAmountIn;
    if (_metarouteTransaction.secondSwapCalldata.length != 0) {
        bytes memory secondSwapCalldata = _metarouteTransaction.secondSwapCalldata;

        assembly {
            mstore(add(secondSwapCalldata, 100), secondSwapAmountIn)
        }

        IERC20(_metarouteTransaction.approvedTokens[approvedTokensLength - 2]).appr
            _metarouteTransaction.secondDexRouter,
            secondSwapAmountIn
        );

        (bool secondSwapSuccess,) = _metarouteTransaction.secondDexRouter.call(
            secondSwapCalldata
        );
        require(secondSwapSuccess, "MetaRouterV2: second swap failed");

        finalSwapAmountIn = IERC20(
            _metarouteTransaction.approvedTokens[approvedTokensLength - 1]
        ).balanceOf(address(this));
    }

    IERC20(_metarouteTransaction.approvedTokens[approvedTokensLength - 1]).approve(
        _metarouteTransaction.relayRecipient,
        finalSwapAmountIn
    );

    bytes memory otherSideCalldata = _metarouteTransaction.otherSideCalldata;
    assembly {
        mstore(add(otherSideCalldata, 100), finalSwapAmountIn)
    }

    (bool otherSideCallSuccess,) = _metarouteTransaction.relayRecipient
                                        .call(otherSideCalldata);
    require(otherSideCallSuccess, "MetaRouterV2: other side call failed");
}
```

Recommendation:

While funds are not expected to remain at rest, it is still advisable to perform approvals only to authorized exchanges and to validate that a swap was indeed made before performing the final transaction. In general, the router should identify the amounts it received via the return arguments of the swaps rather than rely on dynamic `balanceOf` invocations.

Alleviation:

The Symbiosis Finance team stated that given the context of the contract any allowance will not pose a threat to other users or the network and as such they opt to not remediate it.

MRV-03M: Ill-Advised Allowance Pattern

Type	Severity	Location
Standard Conformity	● Minor	MetaRouterV2.sol:L145-L148

Description:

The linked code performs an "infinity" allowance to the router it is meant to interact with, a programming paradigm that is advised against.

Example:

contracts/metarouter/MetaRouterV2.sol

SOL

Copy

```
137 function _swap(
138     address _token,
139     uint256 _amount,
140     address _router,
141     bytes memory _swapCalldata,
142     uint256 _offset
143 ) internal returns (bool success) {
144     if (IERC20(_token).allowance(address(this), _router) < _amount) {
145         IERC20(_token).approve(
146             _router,
147             type(uint256).max
148         );
149     }
150 
151     assembly {
152         mstore(add(_swapCalldata, _offset), _amount)
153     }
154 
155     (success, ) = _router.call(_swapCalldata);
156 }
```

Recommendation:

We advise the allowance to be set to exactly the value necessary to avoid potential complications due to unspent `allowance`.

Alleviation:

The Symbiosis Finance team stated that given the context of the contract any allowance will not pose a threat to other users or the network and as such they opt to not remediate it.

MRV-04M: Improper `receive` Function

Type	Severity	Location
Logical Fault	● Minor	MetaRouterV2.sol:L15

Description:

The `MetaRouterV2` contract is able to receive native assets, however, no function exists in the contract that utilizes funds received as an argument.

Example:

contracts/metarouter/MetaRouterV2.sol

SOL

Copy

```sol
15  receive() external payable {}
```

Recommendation:

Presumably, this function was introduced to allow native outputs in the swaps the contract performs, however, no outward native asset transfer is performed by the contract that utilizes converted or existing (`address(this).balance`) funds. As a result, we advise the function to be omitted from the contract.

Alleviation:

The `receive` function has been omitted from the codebase as per our recommendation.

View Fix on GitHub

Portal Manual Review Findings

ON THIS PAGE

POR-01M: Inexistent Access Control for Reverts

Type	Severity	Location
Logical Fault	● Major	Portal.sol:L373-L412

Description:

The `revertBurnRequest` applies no access control on the caller, allowing arbitrary users to inspect the blockchain mempool and cancel upcoming synthesizes by front-running them with a crafted `_txID`.

Example:

contracts/synth-contracts/Portal.sol

SOL

Copy

```solidity
/**
 * @notice Revert burnSyntheticToken() operation
 * @dev Can called only by bridge after initiation on a second chain
 * @dev Further, this transaction also enters the relay network and is called on th
 * @param _txID the synthesize transaction that was received from the event when it
 * @param _receiveSide Synthesis address on another network
 * @param _oppositeBridge Bridge address on another network
 * @param _chainId Chain id of the network
 */
function revertBurnRequest(
    uint256 _stableBridgingFee,
    bytes32 _txID,
    address _receiveSide,
    address _oppositeBridge,
    uint256 _chainId
) external payable whenNotPaused {
    bytes32 externalID = keccak256(abi.encodePacked(_txID, address(this), block.cha

    require(
    require(
        unsynthesizeStates[externalID] != UnsynthesizeState.Unsynthesized,
        "Symb: Real tokens already transfered"
    );
    unsynthesizeStates[externalID] = UnsynthesizeState.RevertRequest;

    {
        bytes memory out = abi.encodeWithSelector(
            bytes4(keccak256(bytes("revertBurn(uint256,bytes32)"))),
            _stableBridgingFee,
            externalID
        );
        IBridge(bridge).transmitRequestV2(
            out,
            _receiveSide,
            _oppositeBridge,
            _chainId
        );
    }

    emit RevertBurnRequest(_txID, _msgSender());
}
```

Recommendation:

We advise access control to be imposed here properly by allowing the function to only be invoked by the bridge as per the documentation.

Alleviation:

The external ID system now utilizes the `_msgSender()` argument as well thereby ensuring that the ID of a different party cannot be provided and thus alleviating this exhibit.

POR-02M: Improper `receive` Function

Type	Severity	Location
Logical Fault	● Minor	Portal.sol:L585

Description:

The `Portal` contract is able to receive native assets, however, no function exists in the contract that utilizes funds received as an argument.

Example:

contracts/synth-contracts/Portal.sol

SOL

Copy

```sol
585 receive() external payable {}
```

Recommendation:

We advise the function to be omitted from the contract to avoid locked native assets.

Alleviation:

The `receive` function has been omitted from the codebase as per our recommendation.

View Fix on GitHub

POR-03M: Potential of Repeat Invocation

Type	Severity	Location
Logical Fault	● Minor	Portal.sol:L444-L450

Description:

The `setMetaRouter` function can be invoked an arbitrary number of times and set a sensitive contract variable.

Example:

contracts/synth-contracts/Portal.sol

SOL

Copy

```sol
444 /**
445  * @notice Sets MetaRouter address
446  */
447 function setMetaRouter(IMetaRouterV2 _metaRouter) external onlyOwner {
448     require(address(_metaRouter) != address(0), "Symb: metaRouter cannot be zero ad
449     metaRouter = _metaRouter;
450 }
```

Recommendation:

We advise it to only be settable once as otherwise a malicious `owner` can front-run a potential synthesization by setting the `metaRouter` to a malicious contract prior to a transaction's execution by the network.

Alleviation:

The Symbiosis Finance team stated that while they are aware of the power of this feature, they consider it essential to their project and in order to alleviate concerns they will ensure that the owner of the contract will sit behind a multisignature wallet and timelock implementation. As such, we consider this exhibit adequately dealt with.

View Fix on GitHub

SyntERC20 Manual Review Findings

ON THIS PAGE

SER-01M: Arbitrary Burn Operations

Type	Severity	Location
Logical Fault	● Minor	SyntERC20.sol:L16-L18

Description:

The `burn` function of the `SyntERC20` token allows the owner to burn units from an arbitrary account.

Example:

contracts/synth-contracts/SyntERC20.sol

SOL Copy

```sol
function burn(address account, uint256 amount) external onlyOwner {
        _burn(account, amount);
}
```

Recommendation:

We advise a `burnFrom` paradigm to be utilized instead whereby the user has provided sufficient `allowance` to the owner to burn those units to prevent misuse.

Alleviation:

The Symbiosis Finance team stated that the owner will always be the `SyntFabric` contract and as such no arbitrary burn operation can be executed.

Synthesis Manual Review Findings

ON THIS PAGE

SYN-01M: Inexistent Access Control for Reverts

Type	Severity	Location
Logical Fault	● Major	Synthesis.sol:L226-L232

Description:

The `revertSynthesizeRequest` applies no access control on the caller, allowing arbitrary users to inspect the blockchain mempool and cancel upcoming synthesizes by front-running them with a crafted `_txID`.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```sol
217 /**
218  * @notice Revert synthesize() operation
219  * @dev Can called only by bridge after initiation on a second chain
220  * @dev Further, this transaction also enters the relay network and is called on th
221  * @param _txID the synthesize transaction that was received from the event when it
222  * @param _receiveSide Synthesis address on another network
223  * @param _oppositeBridge Bridge address on another network
224  * @param _chainID Chain id of the network
225  */
226 function revertSynthesizeRequest(
227     uint256 _stableBridgingFee,
228     bytes32 _txID,
229     address _receiveSide,
230     address _oppositeBridge,
231     uint256 _chainID,
    uint256 _chainID
) external whenNotPaused {
    bytes32 externalID = keccak256(abi.encodePacked(_txID, address(this), block.cha

    require(
        synthesizeStates[externalID] != SynthesizeState.Synthesized,
        "Symb: synthetic tokens already minted"
    );
    synthesizeStates[externalID] = SynthesizeState.RevertRequest; // close

    {
        bytes memory out = abi.encodeWithSelector(
            bytes4(keccak256(bytes("revertSynthesize(uint256,bytes32)"))),
            _stableBridgingFee,
            externalID
        );
        IBridge(bridge).transmitRequestV2(
            out,
            _receiveSide,
            _oppositeBridge,
            _chainID
        );
    }

    emit RevertSynthesizeRequest(_txID, _msgSender());
}
```

Recommendation:

We advise access control to be imposed here properly by allowing the function to only be invoked by the bridge as per the documentation.

Alleviation:

The external ID system now utilizes the `_msgSender()` argument as well thereby ensuring that the ID of a different party cannot be provided and thus alleviating this exhibit.

SYN-02M: Improper Reversion of Burn

Type	Severity	Location
Logical Fault	● Minor	Synthesis.sol:L394-L423

Description:

The `revertBurn` function does not properly revert the burn action as the `recipient` of the burn operation is not reimbursed for the full amount they burned and instead the bridging fee is applied again.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```solidity
394 /**
395  * @notice Emergency unburn
396  * @dev Can called only by bridge after initiation on a second chain
397  * @param _txID the synthesize transaction that was received from the event when it
398  */
399 function revertBurn(uint256 _stableBridgingFee, bytes32 _txID) external onlyBridge
400     TxState storage txState = requests[_txID];
401     require(
402         txState.state == RequestState.Sent,
403         "Symb: state not open or tx does not exist"
404     );
405     txState.state = RequestState.Reverted;
406     // close
407     ISyntFabric(fabric).synthesize(
408         txState.recipient,
409         txState.amount - _stableBridgingFee,
410         txState.stoken
411     );
412     ISyntFabric(fabric).synthesize(
413         bridge,
414         _stableBridgingFee,
415         txState.stoken
416     );
417     emit RevertBurnCompleted(
418         _txID,
419         txState.recipient,
420         txState.amount,
421         txState.stoken
```

```
    );
}
```

Recommendation:

We advise this trait of the system to be re-evaluated and the bridge fee to potentially not be applied for emergency reversions.

Alleviation:

The Symbiosis Finance team stated that this is indeed by design as the relayers of the cross-chain interaction need to be compensated and will have utilized off-chain resources. As a result, we consider this exhibit null.

View Fix on GitHub

SYN-03M: Inconsistent Event Amount

Type	Severity	Location
Standard Conformity	● Minor	Synthesis.sol:L142, L151, L178, L188, L193

Description:

The `SynthesizeCompleted` event has an inconsistent amount emitted, at one instance emitting the full amount inclusive of the minting fee and at the other emitting the amount sans the fee.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```sol
176 ISyntFabric(fabric).synthesize(
177     address(this),
178     _metaMintTransaction.amount - _metaMintTransaction.stableBridgingFee,
179     syntReprAddr
180 );
181
182 ISyntFabric(fabric).synthesize(
183     bridge,
184     _metaMintTransaction.stableBridgingFee,
185     syntReprAddr
186 );
187
188 _metaMintTransaction.amount = _metaMintTransaction.amount - _metaMintTransaction.st
189
190 emit SynthesizeCompleted(
191     _metaMintTransaction.txID,
192     _metaMintTransaction.to,
193     _metaMintTransaction.amount,
194     _metaMintTransaction.tokenReal
195 );
```

Recommendation:

We advise the event emissions to be synced to ensure that off-chain processes properly process the amounts synthesize, especially in a layer-2 sensitive system such as a bridge.

Alleviation:

The event emissions were standardized to emit the amount sans the fee across the code.

View Fix on GitHub

SYN-04M: Inexistent Validation of Token Existence

Type	Severity	Location
Logical Fault	● Minor	Synthesis.sol:L138, L171-L174

Description:

The linked synthesization lookups do not guarantee that the address exists yet the code assumes so.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```sol
171 address syntReprAddr = ISyntFabric(fabric).getSyntRepresentation(
172     _metaMintTransaction.tokenReal,
173     _metaMintTransaction.chainID
174 );
175 
176 ISyntFabric(fabric).synthesize(
177     address(this),
178     _metaMintTransaction.amount - _metaMintTransaction.stableBridgingFee,
179     syntReprAddr
180 );
```

Recommendation:

We advise this to be evaluated by a proper `require` check to increase code legibility and aid in debugging of the system.

Alleviation:

A `require` check was introduced ensuring that the `syntReprAddr` retrieved is non-zero.

View Fix on GitHub

SYN-05M: Potential of Repeat Invocation

Type	Severity	Location
Logical Fault	● Minor	Synthesis.sol:L448-L454

Description:

The `setMetaRouter` function can be invoked an arbitrary number of times and set a sensitive contract variable.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```sol
448 /**
449  * @notice Sets MetaRouter address
450  */
451 function setMetaRouter(IMetaRouterV2 _metaRouter) external onlyOwner {
452     require(address(_metaRouter) != address(0), "Symb: metaRouter cannot be zero ad
453     metaRouter = _metaRouter;
454 }
```

Recommendation:

We advise it to only be settable once as otherwise a malicious `owner` can front-run a potential synthesization by setting the `metaRouter` to a malicious contract prior to a transaction's execution by the network.

Alleviation:

The Symbiosis Finance team stated that while they are aware of the power of this feature, they consider it essential to their project and in order to alleviate concerns they will ensure that the owner of the contract will sit behind a multisignature wallet and timelock implementation. As such, we consider this exhibit adequately dealt with.

View Fix on GitHub

Wrapper Manual Review Findings

ON THIS PAGE

WRA-01M: Deprecated Native Asset Transfer

Type	Severity	Location
Language Specific	● Minor	Wrapper.sol:L30

Description:

The `transfer` member exposed by `payable` address types has been deprecated as it does not reliably execute and can fail in future updates of the EVM as it forwards a fixed gas stipend which is not compatible with gas cost EIP upgrades such as **EIP-2929**.

Example:

contracts/synth-contracts/utils/Wrapper.sol

```sol
26 function withdraw(uint256 amount) external {
27     address payable payer = payable(_msgSender());
28     require(balanceOf(payer) >= amount);
29     _burn(payer, amount);
30     payer.transfer(amount);
31     emit Withdrawal(payer, amount);
32 }
```

Recommendation:

We advise a safe wrapper library to be utilized instead such as the `sendValue` function of the `Address` library by OpenZeppelin which is guaranteed to execute under all circumstances.

Alleviation:

The Symbiosis Finance team responded by stating this is meant to be used as a test contract and as such they will not carry out any remediations for it.

WRA-02M: Improper `receive` Function

Type	Severity	Location
Logical Fault	● Minor	Wrapper.sol:L19

Description:

The `Wrapper` contract is able to receive native assets, however, no function exists in the contract that utilizes funds received as an argument.

Example:

contracts/synth-contracts/utils/Wrapper.sol

SOL Copy

```sol
19 receive() external payable {}
```

Recommendation:

We advise the function to be omitted from the contract to avoid locked native assets.

Alleviation:

The Symbiosis Finance team responded by stating this is meant to be used as a test contract and as such they will not carry out any remediations for it.

WRA-03M: Inexistent Validation of Amounts

Type	Severity	Location
Input Sanitization	● Minor	Wrapper.sol:L21, L26

Description:

The `deposit` and `withdraw` functions of the contract do not validate that non-zero amounts are being deposited and withdrawn respectively.

Example:

contracts/synth-contracts/utils/Wrapper.sol

SOL

Copy

```solidity
21 function deposit() external payable {
22     _mint(_msgSender(), msg.value);
23     emit Deposit(_msgSender(), msg.value);
24 }
25 
26 function withdraw(uint256 amount) external {
27     address payable payer = payable(_msgSender());
28     require(balanceOf(payer) >= amount);
29     _burn(payer, amount);
30     payer.transfer(amount);
31     emit Withdrawal(payer, amount);
32 }
```

Recommendation:

We advise such sanitization to be imposed to avoid misleading events from being emitted.

Alleviation:

The Symbiosis Finance team responded by stating this is meant to be used as a test contract and as such they will not carry out any remediations for it.

BridgeV2 Code Style Findings

ON THIS PAGE

BV2-01C: Redundant Logical Block

BV2-01C: Redundant Logical Block

Type	Severity	Location
Gas Optimization	Informational	BridgeV2.sol:L63-L69

Description:

The `mpc` function is meant to retrieve either the `newMPC` or `oldMPC` variable depending on the `newMPCEffectiveTime`, however, the variable's value is always set to `block.timestamp` across the contract rendering the check redundant.

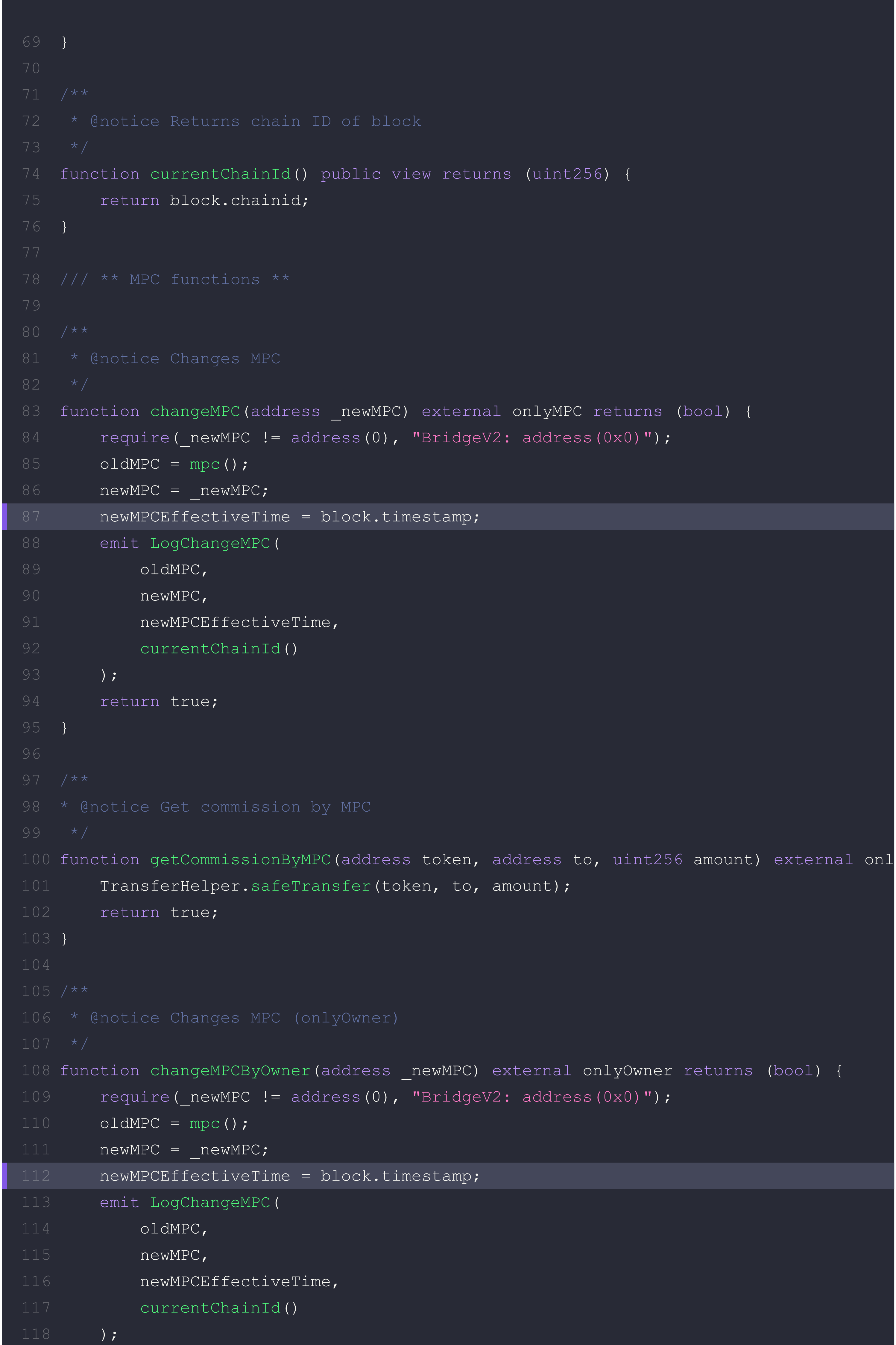
Example:

contracts/synth-contracts/bridge-v2/BridgeV2.sol

SOL

Copy

```sol
49  /// ** INITIALIZER **
50  
51  function initialize(address _mpc) public virtual initializer {
52      __Ownable_init();
53  
54      newMPC = _mpc;
55      newMPCEffectiveTime = block.timestamp;
56  }
57  
58  /// ** VIEW functions **
59  
60  /**
61   * @notice Returns MPC
62   */
63  function mpc() public view returns (address) {
64      if (block.timestamp >= newMPCEffectiveTime) {
65          return newMPC;
66      }
67  
68      return oldMPC;
```



```
}

/**
 * @notice Returns chain ID of block
 */
function currentChainId() public view returns (uint256) {
    return block.chainid;
}

/// ** MPC functions **

/**
 * @notice Changes MPC
 */
function changeMPC(address _newMPC) external onlyMPC returns (bool) {
    require(_newMPC != address(0), "BridgeV2: address(0x0)");
    oldMPC = mpc();
    newMPC = _newMPC;
    newMPCEffectiveTime = block.timestamp;
    emit LogChangeMPC(
        oldMPC,
        newMPC,
        newMPCEffectiveTime,
        currentChainId()
    );
    return true;
}

/**
* @notice Get commission by MPC
 */
function getCommissionByMPC(address token, address to, uint256 amount) external onl
    TransferHelper.safeTransfer(token, to, amount);
    return true;
}

/**
 * @notice Changes MPC (onlyOwner)
 */
function changeMPCByOwner(address _newMPC) external onlyOwner returns (bool) {
    require(_newMPC != address(0), "BridgeV2: address(0x0)");
    oldMPC = mpc();
    newMPC = _newMPC;
    newMPCEffectiveTime = block.timestamp;
    emit LogChangeMPC(
        oldMPC,
        newMPC,
        newMPCEffectiveTime,
        currentChainId()
    );
    return true;
}
```

Recommendation:

We advise the `mpc` function to retrieve `newMPC` directly, optimizing its gas cost.

Alleviation:

The Symbiosis Finance team considered this exhibit but opted not to apply a remediation for it in the current iteration.

MetaRouterV2 Code Style Findings

ON THIS PAGE

MRV-01C: Data Location Optimization

Type	Severity	Location
Gas Optimization	● Informational	MetaRouterV2.sol:L19, L95, L102

Description:

The linked function arguments are set as `memory` yet are declared in `external` functions.

Example:

contracts/metarouter/MetaRouterV2.sol

SOL

Copy

```sol
101 function metaMintSwap(
102     MetaRouteStructs.MetaMintTransaction memory _metaMintTransaction
103 ) external {
```

Recommendation:

We advise them to be set as `calldata` optimizing their read-access gas cost.

Alleviation:

All linked instances were properly adjusted to `calldata`.

MRV-02C: Redundant `constructor` Implementation

Type	Severity	Location
Gas Optimization	● Informational	MetaRouterV2.sol:L16

Description:

The linked `constructor` is redundant.

Example:

contracts/metarouter/MetaRouterV2.sol

SOL

Copy

```sol
16  constructor() public {}
```

Recommendation:

We advise it to be omitted from the codebase.

Alleviation:

The `constructor` function has been omitted from the codebase as per our recommendation.

View Fix on GitHub

Portal Code Style Findings

ON THIS PAGE

POR-01C: Inexistent Error Messages

POR-01C: Inexistent Error Messages

Type	Severity	Location
Code Style	● Informational	Portal.sol:L84, L89

Description:

The linked `require` checks have no error messages explicitly defined.

Example:

contracts/synth-contracts/Portal.sol

SOL

Copy

```sol
89 require(!paused);
```

Recommendation:

We advise them to be set so to aid in the validation of the `require`'s condition as well as the legibility of the codebase.

Alleviation:

Proper error messages were introduced for all linked instances.

RelayRecipientUpgradeable Code Style Findings

ON THIS PAGE

RRU-01C: Redundant Implementation

Type	Severity	Location
Standard Conformity	● Informational	RelayRecipientUpgradeable.sol:L7-L57

Description:

The `RelayRecipientUpgradeable` contract is meant to implement the `ERC2771Context` contract of OpenZeppelin in an upgrade-compatible way, however, the said contract already exists under `metatx/ERC2771ContextUpgradeable` in the `@openzeppelin/contracts-upgradeable` dependency.

Example:

contracts/synth-contracts/RelayRecipientUpgradeable.sol

SOL

Copy

```
5   import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
6
7   abstract contract RelayRecipientUpgradeable is OwnableUpgradeable {
8       address private _trustedForwarder;
9
10      function __RelayRecipient_init(address trustedForwarder)
11          internal
12          initializer
13      {
14          __Ownable_init();
15          _trustedForwarder = trustedForwarder;
16      }
```

Recommendation:

We advise it to be utilized instead as there is no reason the `_trustedForwarder` should be mutable and increases the contract's gas cost.

Alleviation:

The Symbiosis Finance team considered this exhibit but opted not to apply a remediation for it in the current iteration.

RRU-02C: Redundant Import

Type	Severity	Location
Code Style	● Informational	RelayRecipientUpgradeable.sol:L5, L7

Description:

The `RelayRecipientUpgradeable` contract is set as `OwnableUpgradeable` yet none of that contract's traits are utilized.

Example:

contracts/synth-contracts/RelayRecipientUpgradeable.sol

SOL Copy

```sol
5   import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
6
7   abstract contract RelayRecipientUpgradeable is OwnableUpgradeable {
```

Recommendation:

We advise the inheritence to be omitted.

Alleviation:

The Symbiosis Finance team considered this exhibit but opted not to apply a remediation for it in the current iteration.

SyntERC20 Code Style Findings

ON THIS PAGE

SER-01C: Variable Mutability Specifier

Type	Severity	Location
Gas Optimization	Informational	SyntERC20.sol:L10, L27

Description:

The linked variable is assigned to only once during the contract's `constructor`.

Example:

contracts/synth-contracts/SyntERC20.sol

SOL

Copy

```solidity
9   contract SyntERC20 is Ownable, ERC20Permit {
10      uint8 private _decimals;
11
12      function mint(address account, uint256 amount) external onlyOwner {
13          _mint(account, amount);
14      }
15
16      function burn(address account, uint256 amount) external onlyOwner {
17          _burn(account, amount);
18      }
19
20      function decimals() public view virtual override returns (uint8) {
21          return _decimals;
22      }
23
24      constructor(string memory name_, string memory symbol_, uint8 decimals_)
25          ERC20Permit("Symbiosis")
26          ERC20(name_, symbol_) {
27          _decimals = decimals_;
28      }
29  }
```

Recommendation:

We advise it to be set as `immutable` greatly optimizing the codebase.

Alleviation:

The variable has been properly set as `immutable` optimizing the codebase.

SyntFabric Code Style Findings

ON THIS PAGE

SFC-01C: Inexistent Error Messages

Type	Severity	Location
Code Style	● Informational	SyntFabric.sol:L32

Description:

The linked `require` checks have no error messages explicitly defined.

Example:

contracts/synth-contracts/SyntFabric.sol

SOL

Copy

```sol
require(msg.sender == synthesis);
```

Recommendation:

We advise them to be set so to aid in the validation of the `require`'s condition as well as the legibility of the codebase.

Alleviation:

A proper error message was introduced for the linked instance.

SFC-02C: Inexistent Function Implementations

Type	Severity	Location
Code Style	● Informational	SyntFabric.sol:L217, L218

Description:

The code specification mentions functions that are no longer part of the codebase.

Example:

contracts/synth-contracts/SyntFabric.sol

```sol
215 /**
216  * @dev Sets representation
217  * @dev Internal function used in createRepresentationByAdmin, createRepresentation
218  * createRepresentationByTokenOwnerSalted
219  */
```

Recommendation:

We advise the comments to be revised to no longer mention deprecated code.

Alleviation:

The Symbiosis Finance team considered this exhibit but opted not to apply a remediation for it in the current iteration.

Synthesis Code Style Findings

ON THIS PAGE

SYN-01C: Inexistent Error Messages

Type	Severity	Location
Code Style	● Informational	Synthesis.sol:L80, L466

Description:

The linked `require` checks have no error messages explicitly defined.

Example:

contracts/synth-contracts/Synthesis.sol

SOL

Copy

```
80  require(bridge == msg.sender);
```

Recommendation:

We advise them to be set so to aid in the validation of the `require`'s condition as well as the legibility of the codebase.

Alleviation:

Proper error messages were introduced for all linked instances.

Wrapper Code Style Findings

ON THIS PAGE

WRA-01C: Inexistent Visibility Specifier

Type	Severity	Location
Code Style	● Informational	Wrapper.sol:L9

Description:

The linked variable has no visibility specifier explicitly set.

Example:

contracts/synth-contracts/utils/Wrapper.sol

SOL

Copy

```sol
9   address immutable _trustedForwarder;
```

Recommendation:

We advise one to be set so to avoid potential compilation discrepancies in the future as the current behaviour is for the compiler to assign one automatically which may deviate between `pragma` versions.

Alleviation:

The `private` visibility specifier was properly introduced for the linked variable.

Finding Types

ON THIS PAGE

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

External Call Validation

Many contracts that interact with DeFi contain a set of complex external call executions that need to happen in a particular sequence and whose execution is usually taken for granted whereby it is not always the case. External calls should always be validated, either in the form of `require` checks imposed at the contract-level or via more intricate mechanisms such as invoking an external getter-variable and ensuring that it has been properly updated.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted `if` blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Solidity language boasts that discerns it from other conventional programming languages. For example, the EVM is a 256-bit machine meaning that operations on less-than-256-bit types are more costly for the EVM in terms of gas costs, meaning that loops utilizing a `uint8` variable because their limit will never exceed the 8-bit range actually cost more than redundantly using a `uint256` variable.

Code Style

An official Solidity style guide exists that is constantly under development and is adjusted on each new Solidity release, designating how the overall look and feel of a codebase should be. In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a contract-level variable that is present in the inheritance chain of the local execution level's context.

Gas Optimization

Gas optimization findings relate to ways the codebase can be optimized to reduce the gas cost involved with interacting with it to various degrees. These types of findings are completely optional and are pointed out for the benefit of the project's developers.

Standard Conformity

These types of findings relate to incompatibility between a particular standard's implementation and the project's implementation, oftentimes causing significant issues in the usability of the contracts.

Mathematical Operations

In Solidity, math generally behaves differently than other programming languages due to the constraints of the EVM. A prime example of this difference is the truncation of values during a division which in turn leads to loss of precision and can cause systems to behave incorrectly when dealing with percentages and proportion calculations.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.