HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Router Protocol
Date: May 27th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Router Protocol.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Unit and Integration Testing
Platform	EVM
Language	Solidity
Methods	Unit, Functional and Integration Testing
Timeline	09.05.2022 - 27.05.2022
Changelog	27.05.2022 - Testing Report

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Router Protocol (Customer) to conduct an Integrational and Unit Testing Reporting.

Scope

The scope of the project is four repositories containing both smart contracts and system code:

Repositories:
https://github.com/router-protocol/path-finder-api
https://github.com/router-protocol/router-aggregator
https://github.com/router-protocol/router-bridge-contracts-v2
https://github.com/router-protocol/router-bridge
Commits:
2fd7e7eeb2df886943d42fa98dc99ee21ab91b96
ab9c508e71a39f3349d9010536820217a1a39068
45eca63ffb3bca752dfe3e98b1032ce9a7b18ca8
c29a9a8622a139fa39d33c7847b580434dd7919f
Tests Repository:
https://github.com/hknio/router-protocol-tests

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Integration test report

Some code branches were not covered by tests because of different issues which, will be described below. It means the `chains` directory of the router-bridge repository.

chains/ethereum

chain.go (from **72.6%** to **90.4%**)

It is very tricky to force errors.

```
// checkBlockstore queries the blockstore for the latest known block. If the latest block is
// greater than cfg.startBlock, then cfg.startBlock is replaced with the latest known block.
func setupBlockstore(cfg *Config, kp *secp256k1.Keypair) (*blockstore.Blockstore, error) {
        //
        bs, err := blockstore.NewBlockstore(cfg.blockstorePath, cfg.id, kp.Address())
        if err != nil {
                return nil, err
        }

        if !cfg.freshStart {
                latestBlock, err := bs.TryLoadLatestBlock()
                if err != nil {
                        return nil, err
                }
                if latestBlock != nil {
                        if cfg.startBlock != nil {
                                if latestBlock.Cmp(cfg.startBlock) == 1 {
                                        cfg.startBlock = latestBlock
                                }
                        }
                }

        }

        return bs, nil
}
```

```
bridgeContract, err := bridge.NewBridgeUpgradeable(cfg.bridgeContract, conn.Client())
if err != nil {
        return nil, err
}

chainId, err := bridgeContract.FetchChainID(conn.CallOpts())
if err != nil {
        return nil, err
}

if chainId != uint8(chainCfg.Id) {
        return nil, fmt.Errorf("chainId (%d) and configuration chainId (%d) do not match", chainId, chainCfg.Id)
}

erc20HandlerContract, err := erc20Handler.NewERC20HandlerUpgradeable(cfg.erc20HandlerContract, conn.Client())
if err != nil {
        return nil, err
}

// erc721HandlerContract, err := erc721Handler.NewERC721Handler(cfg.erc721HandlerContract, conn.Client())
// if err != nil {
//         return nil, err
// }

genericHandlerContract, err := GenericHandler.NewGenericHandlerUpgradeable(cfg.genericHandlerContract, conn.Client())
if err != nil {
        return nil, err
}

if chainCfg.LatestBlock {
        curr, err := conn.LatestBlock()
        if err != nil {
                return nil, err
        }
        cfg.startBlock = curr
}
```

```
func (c *Chain) Start() error {
        err := c.listener.start()
        if err != nil {
                return err
        }

        err = c.writer.start()
        if err != nil {
                return err
        }

        c.writer.log.Debug("Successfully started chain")
        return nil
}
```

events.go (from **25.0%** to **32.1%**)

```
func (l *listener) handleGenericDepositedEvent(destId msg.ChainId, nonce msg.Nonce) (msg.Message, error) {
        l.log.Info("Handling Generic deposit event", "dest", destId, "nonce", nonce)

        address := l.cfg.genericHandlerContract
        instance, err := GenericHandler.NewGenericHandlerUpgradeable(address, l.conn.Client())
        l.log.Info("Handling generic deposit event")

        Record, err := instance.FetchDepositRecord(&bind.CallOpts{From: l.conn.Keypair().CommonAddress()}, uint8(destId), uint64(nonce))

        _srcChainID, _ := abi.NewType("uint8", "", nil)
        _nonce, _ := abi.NewType("uint64", "", nil)
        _srcAddr, _ := abi.NewType("address", "", nil)
        _destAddr, _ := abi.NewType("address", "", nil)
        _selector, _ := abi.NewType("bytes4", "", nil)
        _data, _ := abi.NewType("bytes", "", nil)
        _hash, _ := abi.NewType("bytes32", "", nil)
        _gas, _ := abi.NewType("uint256", "", nil)
        arguments := abi.Arguments{
                {
                        Type: _srcChainID,
                },
                {
                        Type: _nonce,
                },
                {
                        Type: _srcAddr,
```

1) There is no error handling and ‘arguments. Pack’ panics because ‘Record’ is empty.
2) A deposit record should be created so the test passes.

listener.go (**58.3%**)

Not possible to test it. The existing test fails, and the current implementation needs to be refactored.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

proposal_data.go (from **0.0%** to **100.0%**)

writer.go (**36.4%**)

```
// ResolveMessage handles any given message based on type
// A bool is returned to indicate failure/success, this should be ignored except for within tests.
func (w *writer) ResolveMessage(m msg.Message) bool {
	w.log.Info("Attempting to resolve message", "type", m.Type, "src", m.Source, "dst", m.Destination, "nonce", m.DepositNonce, "rId", m.ResourceId.Hex())

	switch m.Type {
	case msg.FungibleTransfer:
		return w.createErc20Proposal(m)
	case msg.NonFungibleTransfer:
		return w.createErc721Proposal(m)
	case msg.GenericTransfer:
		return w.createGenericDepositProposal(m)
	default:
		w.log.Error("Unknown message type received", "type", m.Type)
		return false
	}
}
```

Not possible to test because ‘TestWriter_ResolveMessage’ itself is not working. As soon as it is fixed, this code block will be invoked.

chains/substrate

chain.go (from **71.7%** to **91.3%**)

Tricky to force errors.

```
// checkBlockstore queries the blockstore for the latest known block. If the latest block is
// greater than startBlock, then the latest block is returned, otherwise startBlock is.
func checkBlockstore(bs *blockstore.Blockstore, startBlock uint64) (uint64, error) {
	latestBlock, err := bs.TryLoadLatestBlock()
	if err != nil {
		return 0, err
	}

	if latestBlock.Uint64() > startBlock {
		return latestBlock.Uint64(), nil
	} else {
		return startBlock, nil
	}
}
```

```
	startBlock := parseStartBlock(cfg)
	if !cfg.FreshStart {
		startBlock, err = checkBlockstore(bs, startBlock)
		if err != nil {
			return nil, err
		}
	}
```

```
        if cfg.LatestBlock {
                curr, err := conn.api.RPC.Chain.GetHeaderLatest()
                if err != nil {
                        return nil, err
                }
                startBlock = uint64(curr.Number)
        }

        ue := parseUseExtended(cfg)

        // Setup listener & writer
        l := NewListener(conn, cfg.Name, cfg.Id, startBlock, logger, bs, stop, sysErr, m)
        w := NewWriter(conn, logger, sysErr, m, ue)
        return &Chain{
                cfg:      cfg,
                conn:     conn,
                listener: l,
                writer:   w,
                stop:     stop,
        }, nil
}

func (c *Chain) Start() error {
        err := c.listener.start()
        if err != nil {
                return err
        }
        c.conn.log.Debug("Successfully started chain", "chainId", c.cfg.Id)
        return nil
}
```

connection.go (**70.8%**)

Tricky to force errors.

events.go (from **0.0%** to **100.0%**)

listener.go (**28.2%**)

Not possible to test it. The existing test fails, and the current implementation needs to be refactored.

types.go (**0.0%**)

Not possible to test it. A lot of external dependencies that are hard to force errors.

writer.go (**1.6%**)

Not possible to test it. Existing tests are failing due to an existing issue that needs to be further investigated.

Integration test conclusions

The project itself does not have real unit tests. All of them are integration ones. Unit test is meant to test a given unit in isolation,

mocking up any dependency. It is not possible right now due to the way the code base is structured.

To enable unit testing, rely on abstractions rather than concrete types, so any dependencies can be mocked.

More integration tests have been added and raised test coverage for files under the `chains` folder. Some could not write tests because existing issues running them locally.

Integration test executions

To run the tests for `chains/ethereum` files, run:

```
go test -coverprofile=cover_test.out -v
github.com/router-protocol/router-bridge/chains/ethereum
```

Generate the report so that it can be visualized in a browser:

```
go tool cover -html=cover_test.out -o
../coverage_test_ethereum.html
```

To run the tests for `chains/substrate` files, run:

```
go test -coverprofile=cover_test.out -v
github.com/router-protocol/router-bridge/chains/substrate
```

Generate the report so that it can be visualized in a browser:

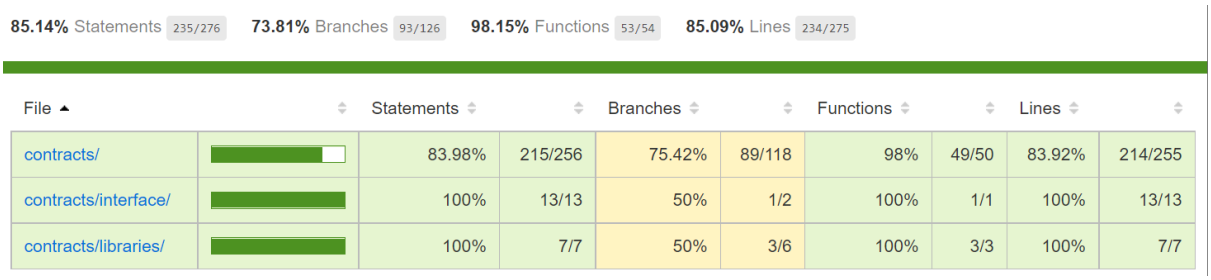
```
go tool cover -html=cover_test.out -o
../coverage_test_substrate.html
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Unit testing for smart contracts

router-aggregator

The current coverage:



85.14% Statements 235/276 **73.81%** Branches 93/126 **98.15%** Functions 53/54 **85.09%** Lines 234/275

File		Statements		Branches		Functions		Lines	
contracts/		83.98%	215/256	75.42%	89/118	98%	49/50	83.92%	214/255
contracts/interface/		100%	13/13	50%	1/2	100%	1/1	100%	13/13
contracts/libraries/		100%	7/7	50%	3/6	100%	3/3	100%	7/7

Total number of successful tests: **70**
Total number of errors: **27**

which are:

- **23** - issues of using `Factory's` constants in smart contracts.
- **3** - issues of using constants for `IWETH` in the `OneSplitRoot smart contract.`

An example of using constants of the `Factory` and the `IWETH` in the `OneSplitRoot smart contract`:

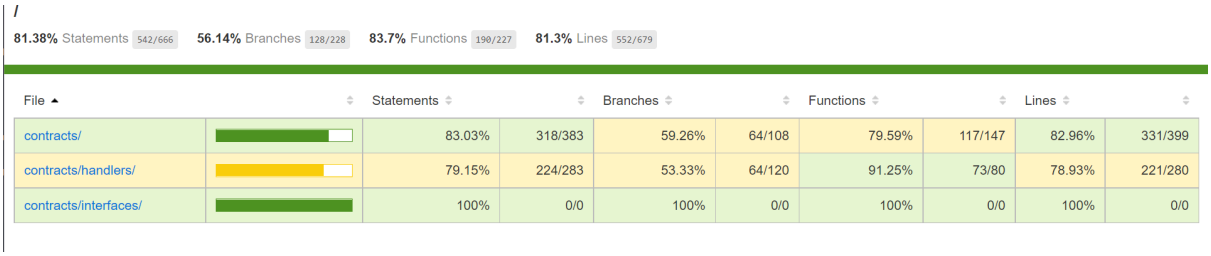
```
IUniswapV2Factory internal constant uniswapV2 = IUniswapV2Factory(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f);
IUniswapV2Factory internal constant dfynExchange = IUniswapV2Factory(0x8BCeDD62DD46F1A76F8A1633d4f5B76e0CDa521E);
IUniswapV2Factory internal constant pancakeSwap = IUniswapV2Factory(0xEF45d134b73241eDa7703fa787148D9C9F4950b0);
IUniswapV2Factory internal constant quickSwap = IUniswapV2Factory(0x9Ad6C38BE94206cA50bb0d90783181662f0Cfa10);
IUniswapV2Factory internal constant sushiSwap = IUniswapV2Factory(0xc35DADB65012eC5796536bD9864eD8773aBc74C4);

IERC20Upgradeable internal constant NATIVE_ADDRESS = IERC20Upgradeable(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
IWETH internal constant wnative = IWETH(0x4200000000000000000000000000000000000006);

int256 internal constant VERY_NEGATIVE_VALUE = -1e72;
address internal constant skimAddress = 0xb599a1e294Ec6608d68987fC9A2d4d155eEd9160;
```

router_router-bridge-contracts-v2

The current coverage:



/
81.38% Statements 542/666 **56.14%** Branches 128/228 **83.7%** Functions 190/227 **81.3%** Lines 552/679

File		Statements		Branches		Functions		Lines	
contracts/		83.03%	318/383	59.26%	64/108	79.59%	117/147	82.96%	331/399
contracts/handlers/		79.15%	224/283	53.33%	64/120	91.25%	73/80	78.93%	221/280
contracts/interfaces/		100%	0/0	100%	0/0	100%	0/0	100%	0/0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Total number of successful tests: **123.**
Total number of errors: **14.**

which are:

- 2 - incorrect smart contract implementation.
- 9 - errors in E2E-testing.
- 3 - undefined issues.

While testing has been founded the following issues:

1. Incorrect smart contract implementation. It is impossible to test some methods of the **`HandlerHelpersUpgradeable`** contract without calling **`setReserve`** method of the class-inheritor **`IHandlerReserve`**:

```
function setLiquidityPoolOwner(
    address newOwner,
    address tokenAddress,
    address lpAddress
) public virtual override onlyRole(BRIDGE_ROLE) {
    _reserve._setLiquidityPoolOwner(newOwner, tokenAddress, lpAddress);
}
```

2. Incorrect smart contract implementation. The limit of the chainId as uint8 (see https://chainlist.org/). As the result — an error: **`value out-of-bounds for uint8`**
3. **`RouterERC20Upgradable.sol`** is using the constructor, which is incorrect for **`Upgradable`** contracts. That led to deploying without proxying for testing.
4. The **`RouterERC20UpgradableOld.sol`** has the same contract name as the **`RouterERC20Upgradable.sol`** has too.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

router_path-finder-api

The current coverage:

86.21% Statements 75/87 **75%** Branches 36/48 **100%** Functions 16/16 **89.16%** Lines 74/83

File		Statements		Branches		Functions		Lines	
contracts/		86.21%	75/87	75%	36/48	100%	16/16	89.16%	74/83

Total number of successful tests: 31.
Total number of errors: 6.

which are:

- 6 - issues of using constants in smart contracts.

```
contract FetchLiquidity {
    using SafeMath for uint256;
    using DisableFlags for uint256;

    IUniswapV2Factory internal constant dfynExchange =
        IUniswapV2Factory(0xd9820a17053d6314B20642E465a84Bf01a3D64f5);
    IUniswapV2Factory internal constant uniswapV2 =
        IUniswapV2Factory(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f);
    IUniswapV2Factory internal constant pancakeSwap =
        IUniswapV2Factory(0xEF45d134b73241eDa7703fa787148D9C9F4950b0);
    IUniswapV2Factory internal constant quickSwap =
        IUniswapV2Factory(0x152eE697f2E276fA89E96742e9bB9aB1F2E61bE3);
    IUniswapV2Factory internal constant sushiSwap =
        IUniswapV2Factory(0xc35DADB65012eC5796536bD9864eD8773aBc74C4);
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Unit test executions

To run get the coverage results, it needs to run

```
yarn coverage
```

in the corresponding repository.

The coverage results would be in the HTML file located at: `**coverage/index.html**`

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusions

The current tests coverage could be improved, and the number of “red-tests” could be decreased by modifying the current smart contracts code.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.