Smart Contract Security Audit Report

Uno Re

1. Contents

2. General Information

This report contains information about the results of the security audit of the Uno Re (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 11/03/2024 to 20/03/2024.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the modifications of the contracts in the repository: https://github.com/Uno-Re/SSIP-SSRP-contracts from the commit 64a9895 up to e3ace21.

The following contracts have been tested:

1. contracts/EIP712MetaTransaction.sol
2. contracts/factories/SyntheticSSRPFactory.sol
3. contracts/factories/SalesPolicyFactory.sol
4. contracts/factories/RiskPoolFactory.sol
5. contracts/factories/RewarderFactory.sol
6. contracts/factories/SyntheticSSIPFactory.sol
7. contracts/RiskPoolERC20.sol
8. contracts/SingleSidedReinsurancePool.sol
9. contracts/libraries/EIP712Base.sol
10. contracts/libraries/TransferHelper.sol
11. contracts/libraries/MultiSigWallet.sol

12. contracts/ExchangeAgent.sol
13. contracts/SingleSidedInsurancePool.sol
14. contracts/CapitalAgent.sol
15. contracts/CapitalAgent1.sol
16. contracts/uma/EscalationManager.sol
17. contracts/uma/RequestPayout.sol
18. contracts/RiskPool.sol
19. contracts/Rewarder.sol
20. contracts/SalesPolicy.sol
21. contracts/PremiumPool.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, policy owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- Insurance policy owners,
- Claim assessors,
- Claim processors,
- UMA disputers.

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 several medium 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). Also, a large portion of the code was not functioning properly, and the test coverage was low at the beginning of the review. We've reported this to Uno Re and the code and tests were partially fixed.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of March 20, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Sender is not properly checked when buying a policy	contracts/SalesPolicy.sol	**Medium**	Not fixed
Users can harvest rewards during emergencyWithdraw mode	contracts/SingleSidedReinsurancePool.sol contracts/SingleSidedInsurancePool.sol	**Medium**	Not fixed
Tokens with approval race protection are incompatible with ExchangeAgent	contracts/uma/PayoutRequest.sol contracts/ExchangeAgent.sol	**Medium**	Not fixed
Default 5% slippage allows to frontrun calls to convertForToken() and convertForETH()	contracts/ExchangeAgent.sol	**Low**	Not fixed

Issue	Contract	Risk Level	Status
Unnecessary `msgSender` function usage	contracts/SalesPolicy.sol	**Info**	Not 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:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Sender is not properly checked when buying a policy

Risk Level: Medium

Status: Not fixed

Contracts:

- contracts/SalesPolicy.sol

Location: Lines: 154. Function: buyPolicy.

Description:

In the buyPolicy() function the msg.sender value is used in getSender() as sender argument:

```
contracts/SalesPolicy.sol:
  143:         address _signer = getSender(
  144:             _policyPriceInUSDC,
  145:             _protocols,
  146:             _coverageDuration,
  147:             _coverageAmount,
  148:             _signedTime,
  149:             _premiumCurrency,
  150:             r,
  151:             s,
  152:             v,
  153:             nonce,
  154:             msg.sender //@audit should be msgSender()
  155:         );
```

In case it's a EIP721MetaTransaction call via executeMetaTransaction(), msg.sender will be equal to the SalesPolicy contract address itself, which means that the sender of the signature is not a part of signed data.

Remediation:

Consider using msgSender() instead of msg.sender in sender argument when calling getSender(), so that the actual sender address is signed.

5.2. Users can harvest rewards during emergencyWithdraw mode

Risk Level: Medium

Status: Not fixed

Contracts:

- contracts/SingleSidedReinsurancePool.sol
- contracts/SingleSidedInsurancePool.sol

Description:

When emergency mode is active, users can withdraw their deposits via `emergencyWithdraw`, however they can call `harvest` before and collect their rewards. This will be equivalent to the normal withdraw of funds, however it will allow users to bypass 10 day lock.

Remediation:

Consider checking that emergency mode is not enabled in `_harvest` function in order to not allow claiming rewards during emergency withdraw period.

5.3. Tokens with approval race protection are incompatible with ExchangeAgent

Risk Level: Medium

Status: Not fixed

Contracts:

- contracts/uma/PayoutRequest.sol
- contracts/ExchangeAgent.sol

Description:

Some tokens, for example USDT and KNC have approval race protection mechanism and require the allowance to be either 0 or `uint256.max` when it is updated. The problem is

that ExchangeAgent uses safeApprove from Uniswap's TransferHelper and PayoutRequest is using raw approve function call which will revert on such tokens:

```
contracts/uma/PayoutRequest.sol:
  84:             defaultCurrency.approve(address(optimisticOracle), bond);

contracts/ExchangeAgent.sol:
  233:             TransferHelper.safeApprove(_token0, address(_dexRouter),
_convertAmount);
```

Remediation:

Make sure to use forceApprove from SafeERC20.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol

5.4. Default 5% slippage allows to frontrun calls to convertForToken() and convertForETH()

Risk Level: Low

Status: Not fixed

Contracts:

- contracts/ExchangeAgent.sol

Location: Lines: 73.

Description:

Default 5% swap slippage that is set in the ExchangeAgent constructor in the storage variable slippage is too high. There is a possibility of the frontrunning calls to the convertForToken() and convertForETH() swap functions which can lead to the loss of swapped tokens in the amount of 5% of the swap.

Remediation:

Reduce the default slippage. The amount of slippage should be based on the sandwich marginal risk and the liquidity in the pool.

5.5. Unnecessary `msgSender` function usage

Risk Level: Info

Status: Not fixed

Contracts:

- contracts/SalesPolicy.sol

Description:

There are two modifiers in the `SalesPolicy` contract that check access control for the caller and they are both using `msgSender()` function from the inherited `EIP712MetaTransaction` contract.

```
contracts/SalesPolicy.sol:
   92      modifier onlyFactory() {
   93:         require(msgSender() == factory, "UnoRe: SalesPolicy
Forbidden");
   94          _;

   97      modifier onlyCapitalAgent() {
   98:         require(msgSender() == capitalAgent, "UnoRe: SalesPolicy
Forbidden");
   99          _;
```

Because both contracts the `SalesPolicyFactory` and the `capitalAgent` can call `SalesPolicy` only directly, it is unnecessary to use `msgSender()` for this purpose.

Remediation:

Consider using `msg.sender` instead of `msgSender()`.

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.