SANCTUM SECURITY

Project: UNION Protocol Foundation Collateral Optimization (C-OP) Smart Contract Audit

Final Security Assessment Report

April 21st, 2021

Table of Contents

Disclaimer

Sanctum Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technology proprietors, business, business model or legal compliance. Sanctum Reports represent an extensive auditing process intending to help you increase the quality of the provided code, while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

All labels, verification services, and work product is provided “AS IS” and Sanctum hereby disclaims all warranties, whether express, implied, statutory, or otherwise and Sanctum specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement, and all warranties arising from course of dealing, usage, or trade practice. Without limiting the foregoing, Sanctum makes no warranty of any kind that the labels, the verification report or work product, or any products or results of the use thereof, will meet your or any other person’s requirements, operate without interruption, achieve any intended result, be compatible or work with any software, system, or other services, or be secure, accurate, complete, free of harmful code, or error-free.

You are responsible for you own due diligence and continuous security. Sanctum is an aid to help you reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Overview

Type: Decentralized Finance (DeFi)

Virtual Machine: Ethereum Virtual Machine

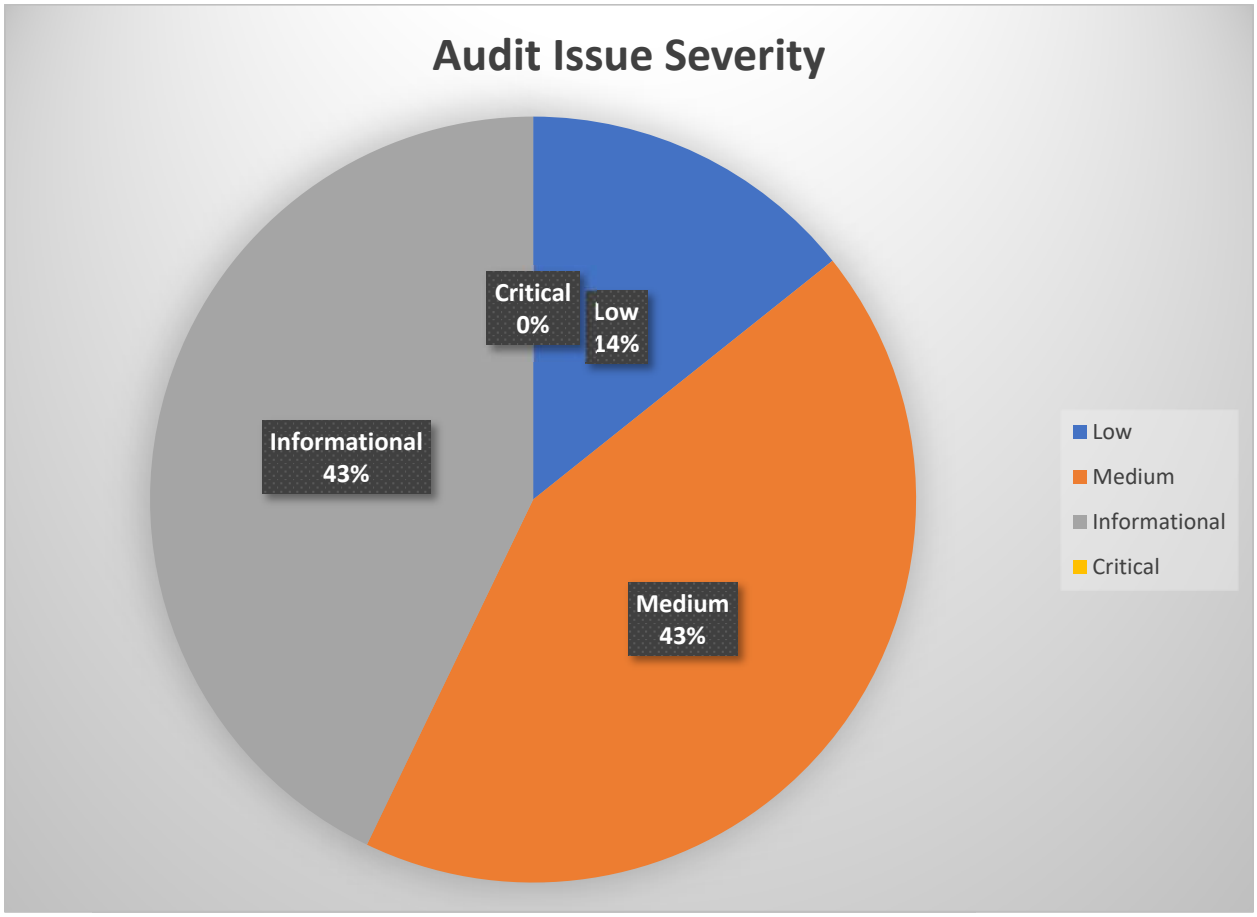
Repository: `https://github.com/UNIONProtocolFoundation/union-protocol-oc-protecton`

Repository Branch Signature: `90429ecd7f0229ab47773c7876ac3bc0fabdbaaa`

Timeframe: April 21, 2021.

Versions:

1. UNION Protocol Foundation C-OP Smart Contract Audit – Final Security Assessment Report (April 21, 2021)



Summary

Code was manually reviewed and analyzed with static analysis tools. 7 findings were identified during the engagement - 5 of them were confirmed by the development team and fixed in the latest update.

All security concerns were investigated by auditors, these risks are low because protection mechanisms are presented for all of them.

The code is clean and well-written; no residual vulnerabilities were identified.

Items Addressed

The following contracts were audited as part of this security review. The branch of the UNION Protocol Foundation repository was given as: `90429ecd7f0229ab47773c7876ac3bc0fabdbaaa`.

1. contracts/AggregatorV3Interface.sol
2. contracts/datatypes/StructuredLinkedList.sol
3. contracts/interfaces
   - IERC1643.sol
   - IERC20Token.sol
   - IOCProtectionSeller.sol
   - IOCProtectionStorage.sol
   - IParamStorage.sol
   - IPool.sol
   - IProtection.sol
   - IUUNNRegistry.sol
   - IUnionRouter.sol
4. contracts/libraries/SignLib.sol
5. contracts/pool/PoolUpgradable.sol
6. contracts/test
   - TestList.sol
   - TestSign.sol
   - TestToken.sol
   - UniswapUtil.sol
7. contracts/upgradable
   - util/Address.sol
   - util/Context.sol
   - util/Ownable.sol
   - BeaconProxy.sol
   - IBeacon.sol
   - Proxy.sol
   - ProxyAdmin.sol
   - TransparentUpgradeableProxy.sol
   - UpgradeableBeacon.sol
   - UpgradeableProxy.sol
8. Migrations.sol
9. OCProtections.sol
10. ParamStorage.sol
11. UnionAssetPool.sol
12. UnionERC20Pool.sol
13. UnionRouter.sol
14. uUNNToken.sol

The audit of these contracts was completed April 21, 2021.

Security Risks

The UNION team described 4 main concerns/risks for the project:

1. Risks from flash loans and composable use
2. Gas and performance
3. Maintenance concerns
4. Correct guarding and permissions around minting of pUNN and uUNN tokens

Risks from flash loans and composable use

ChainLink is used as an oracle for price feeds. It's considered secure and doesn't allow price manipulations within the same block. Moreover, Union Protocol architecture requires server interaction to buy the protection, which creates an extra layer of defense from flash-loan-style risks. If the price is dumped for several blocks - it's expected behavior is to execute the option.

Gas and performance

Smart contracts don't contain heavy loops. Contracts have rich logic thus will be expensive anyway. From the DoS via gas limits perspective - no vulnerabilities were found.

Some gas optimization could be made by changing functions from public to external (Finding 6), but the development team decided to keep them public.

Maintenance concerns

Contracts are upgradable and pausable that may help in case of any security incident. Roles functionality is straightforward and no issues were identified for the functionality of the role.

Correct guarding and permissions around minting of pUNN and uUNN tokens

uUNN can be minted only by `PROTECTION_PREMIUM_DATA_PROVIDER`, if it's trusted - everything's good. uUNN is minted within the `createTo` function.

pUNN is minted only minted `in_deposit` function. It mints the exact amount of pUNN as a basic token. As far as the basic token is limited, pUNN will be limited to the same numbers.

Summary of Issues

Issue #	Description	Issue Severity
1	No access control for documents creation and deletion	Medium
2	ProtectionUpgradable and OCProtections both implement documents functionality	Medium
3	Wrong premium comparison condition in OCProtections	Medium
4	ProtectionSeller.sol is never used	Low
5	Commented pieces of code	Informational
6	Usage of public functions instead of external	Informational
7	Code not fully covered with in-comments documentation	Informational

Security Assessment Details

Issue #1: No access control for documents creation and deletion

Severity: Medium

Issue Description: In `ProtectionUpgradable.sol and OCProtections.sol`, arbitrary accounts were able to access privileged functionality - to create and delete documents..

Recommended Action(s): `onlyOwner` should be able to create and delete documents

Final Status: Document functionality was removed from the code of `OCProtections.sol`; `ProtectionUpgradable.sol` file was removed. The issue was fixed.

Issue #2: ProtectionUpgradable and OCProtections both implement documents functionality

Severity: Medium

Issue Description: In `ProtectionUpgradable.sol and OCProtections.sol`, duplicated functionality creates a lot of issues with data integrity. It also may affect gas consumption for the system.

Recommended Action(s): Document functionality should be implemented in one file or entirely removed.

Final Status: Document functionality was removed from the code of `OCProtections.sol`; `ProtectionUpgradable.sol` file was removed. The issue was fixed.

Issue #3: Wrong premium comparison condition in OCProtections.sol

Severity: Medium

Issue Description: In OCProtections.sol, if there is enough premium but `protections[_id].premium != _premium` function will revert. It violates business logic of the contract.

Recommended Action(s):

```
require (protections[_id].premium <= _premium, "Not enough premium left");
```

Should be changed to

```
require (protections[_id].premium >= _premium, "Not enough premium left");
```

Final Status: Recommended fix was applied. The issue was fixed.

Issue #4: ProtectionSeller.sol is never used

Severity: Low

Issue Description: In `ProtectionSeller.sol` Unused code creates an extra attack surface for the attacker; violates smart contract development best practice and makes the code more difficult to understand.

Recommended Action(s): ProtectionSeller.sol should be removed

Final Status: Recommended fix was applied. The issue was fixed.

Issue #5: Commented pieces of code

Severity: Informational (Best Practice)

Issue Description: It is a best practice that the final code doesn't contain pieces of code in the comments. It may give additional knowledge to the attacker or force the developer to make a mistake and uncomment insecure functionality.

Recommended Action(s): All code in the comments should be removed.

Final Status: Recommended fix was applied. The issue was fixed.

Issue #6: Usage of public functions instead of external

Severity: Informational (Best Practice)

Issue Description: In all files, Some functions are public, not external. For example, createTo would utilize much less gas if it's external. Note. In a nutshell, public and external differs in terms of gas usage. The former use more than the latter when used with large arrays of data. This is because Solidity copies arguments to memory on a public function while external read from calldata which is cheaper than memory allocation.

Recommended Action(s): All functions that are not required to be public should be changed to external .

Final Status: The issue does not have a security impact. The risk was accepted by the customer.

Issue #7: Code not fully covered with in-comments documentation.

Severity: Informational (Best Practice)

Issue Description: In all files, it's a best practice to fully cover the code with documentation. It increases code quality and reduces the risks of business logic issues.

Recommended Action(s): It's highly recommended adding in-comment documentation for all functions.

Final Status: The issue does not have a security impact. The risk was accepted by the customer.
