Report for:

Float Protocol

April 2021

Version: 1.0

Prepared By: Extropy.IO
Email: info@extropy.io
Telephone: +44 1865261424

Executive Summary

This report presents the findings of the security assessment conducted on behalf of Float Protocol. The assessment was conducted between 21/04/21 and 04/05/2021 and was authorised by Float Protocol.

Overview

The project was of high quality, consequently we have only Informational issues to report. The code followed best practices and was appropriate to its purpose.
There was clearly a testing framework in place, complemented by the use of mock contracts.

RETESTED 1 May 2021
The recommendations were followed and all issues fixed or dismissed as false positives. The smart contracts follow good design practices and are robust.

The following table breaks down the issues which were identified by phase and severity of risk.

Phase	Description	Critical	High	Medium	Low	Info	Total
1	Smart Contracts Audit	0	0	0	0	8	9
2	Re test	0	0	0	0	0	0

Assessment Summary

All issues reported here are reported for information only, nevertheless, it was recommended that these be reviewed and addressed , all of which was done.

More detailed information on each of the issues which were identified is included in Section 2.

Table of Contents

Using This Report

To facilitate the dissemination of the information within this report throughout your organisation, this document has been divided into the following clearly marked and separable sections.

Document Breakdown		
0	Executive Summary	Management level, strategic overview of the assessment and the risks posed to the business
1	Technical Summary	An overview of the assessment from a more technical perspective, including a defined scope and any caveats which may apply
2	Technical Details	Detailed discussion (including evidence and recommendations) for each individual security issue which was identified
3	Supplemental Data	Any additional evidence which was too lengthy to include in Section 2
4	Appendices	This section usually includes the security tools which were used, outlines the assessment methodologies and lists the assessment team members

Document Control

Document History			
Issue No.	**Issue Date**	**Issued By**	**Change Description**
0.1	30/04/2021	Laurence Kirk	Draft for Extropy internal review only
1.0	04/05/2021	Laurence Kirk	Released to client

Document Distribution List	
Development team	Float Protocol
Laurence Kirk	CEO, Extropy

1. Technical Summary.

Extropy was contracted by Float Protocol to conduct a Smart Contracts vulnerability assessment and code review in order to identify security issues that could negatively affect Float Protocol's business or reputation if they led to the compromise or abuse of systems.

1.1 Scope

The scope of the audit was the code contained in the primary-protocol branch of the repo, with the following focus.

Primary

- contracts/auction/*
- contracts/funds/*
- contracts/policy/*
- contracts/lib/*

Secondary

- contracts/oracle/*
- contracts/tokens/*

Out of scope:

- contracts/external-lib
- contracts/staking/

The Audit was conducted from commit **521f9c79f32b4ad66bf6192d72a5229d75c6ccba** to commit **bfcb47b228ff8ef65d942bba26fb8b346912395b**

Design

Our understanding of the design of the Ethereum smart contracts in scope for this assessment is as follows

The FLOAT token is a stablecoin whose value can change over time in response to demand. Expansion and contraction in supply are handled by auctions, and an additional token BANK is used as a governance token.

1.2 Disclaimer

The audit makes no statements or warranty about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2 Technical Findings – Smart Contracts Audit

The remainder of this document is technical in nature and provides additional detail about the items already discussed, for the purposes of remediation and risk assessment.

2.1 Check function arguments for valid values

Check function arguments for valid values	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

The values passed to functions should be checked for validity

```
function setTargetPrice(uint256 _targetPrice)
```

Recommendation:

Use a require statement to ensure on valid values are accepted by a function.

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
MonetaryPolicyV1.sol

2.2 Use a consistent compiler version

Use a consistent compiler version	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

A fixed rather than floating compiler version that is consistent between contracts should be used.

```
pragma solidity >=0.5.0;
```

Recommendation:

Fix the compiler version with pragma solidity =0.7.6; in all contracts

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
Math.sol

References :https://consensys.net/blog/developers/solidity-best-practices-for-smart-contract-security/

2.3 State variable visibility

State variables can be made immutable or internal	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

For optimisation, ff a getter function is not needed, it is recommended to declare state variables as immutable or internal.

```
ISupplyControlledERC20 public override float;
```

Recommendation:

Declare variables as internal or immutable where suitable

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
AuctionHouse.sol
Staged.sol

References : https://medium.com/coinmonks/gas-optimization-in-solidity-part-i-variables-9d5775e43dde

2.4 Function Visibility

Change visibility from public to external	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

Marking functions as external rather than public reduces gas cost. Arguments in external functions use call data rather than memory, which is a cheaper allocation.

```
function updateStage() public timedTransition returns (Stages)
```

Recommendation:

Prefer external over public whenever possible.

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
ChainlinkEthUsdConsumer.sol
Staged.sol

2.5 Data type size

Modifier Needed	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

The cost of allocation of storage variables depends on their size.

```
uint256 _auctionCooldown
```

Recommendation:

Use the smallest size possible for variables and constants

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
AuctionHouse.sol

2.6 Default Variable Assignment

Modifier Needed	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

It is unnecessary to set variables to their default value

```
uint256 public round = 0;
```

Recommendation:

Remove the assignment

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
AuctionHouse.sol

2.7 Consistent code style

Use a consistent code style	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

Differing code style and syntax can achieve the same results.

Recommendation:

For readability it is recommended to use a consistent style.
For example

- Where a return variable is declared in the function signature, do not use a return statement in the function.
- Use an explicit size for unsigned integer data types.

Re tested 01 May 2021

- Recommendations followed

Affects:

Smart Contract
Staged.sol
`Math.sol`

2.8 Unneeded Library Call

Unneeded Library Call	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational

Description:

Where overflow is infeasible, a call to the SafeMath library is unnecessary

```
round = round.add(1);
```

Recommendation:

Use the increment operator to save gas.

Re tested 01 May 2021

- Recommendation followed

Affects:

Smart Contract
AuctionHouse.sol

2.9 Tool List

The following tools were used during the assessment:

Tools Used	Description	Resources
Slither	Static analysis	https://github.com/crytic/slither
Surya	Code visualizer	https://github.com/ConsenSys/surya
SWC Registry	Vulnerability database	https://swcregistry.io/

3 Tailored Methodologies

3.1 Smart Contracts Audit

3.1.1 Audit Goals

We will audit the code in accordance with the following criteria:

Sound Architecture

This audit includes assessments of the overall architecture and design choices. Given the subjective nature of these assessments, it will be up to the development team to determine whether any changes should be made.

Smart Contract Best Practices

This audit will evaluate whether the codebase follows the current established best practices for smart contract development.

Code Correctness

This audit will evaluate whether the code does what it is intended to do.

Code Quality

This audit will evaluate whether the code has been written in a way that ensures readability and maintainability.

Security

This audit will look for any exploitable security vulnerabilities, or other potential threats .

Testing and testability

This audit will examine how easily tested the code is, and review how thoroughly tested the code is.

Although we have commented on the application design, issues of crypto-economics, game theory and suitability for business purpose as they relate to this project are beyond the scope of this audit.

3.2 Test Methodology

The security audit is performed in two phases:

Independent Code Review

The code is inspected separately by two team members checking for software errors and known vulnerabilities.

Static Analysis

The code is subject to static analysis as specified above.