TrustToken

Ragnarok

SMART CONTRACT AUDIT

20.01.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, 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.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of TrustToken Inc. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (10.01.2022)	Layout
0.2 (14.01.2022)	Test Deployment
0.5 (17.01.2022)	Manual & Automated Security Testing
0.6 (18.01.2022)	Testing SWC Checks
0.7 (19.01.2022)	Verify Claims
0.9 (20.01.2022)	Summary and Recommendation
1.0 (20.01.2022)	Final document
1.1 (TBA)	Added deployed contract addresses

2. About the Project and Company

Company address:

TrustToken Inc.
234 S Main Street Suite 7 Willits
California 95490
United States of America

Website: https://www.trusttoken.com

Twitter: https://twitter.com/TrustToken

Reddit: https://www.reddit.com/r/TrustToken

Telegram: https://t.me/jointruefi

Discord: https://bit.ly/chattruefi

LinkedIn: https://www.linkedin.com/company/trusttoken

Facebook: https://www.facebook.com/TrustToken/

Medium: https://trusttokenteam.medium.com

YouTube: https://www.youtube.com/channel/UCePpU7NPWENI6rdmFb7HALA

2.1 Project Overview

TrustToken is a platform to create asset-backed tokens that you can easily buy and sell around the world. For example, gold to gold tokens or dollar to dollar tokens. The company's first asset token is TrueUSD, a stablecoin that you can redeem 1-for-1 for US dollars. TrustToken was founded in 2017 and consists of a team from Stanford, UC Berkeley,Airbnb, Goldman Sachs, PayPal, and Google, and is backed by a16z crypto, BlockTower Capital, Danhua Capital, Founders Fund Angel, GGV Capital, Jump Capital, Stanford-StartX, and others.

TrustToken has launched TrueFi, the protocol for uncollateralized lending, powered by the first ever on-chain credit scores and governed by holders of the TRU token. At launch on November 21st, 2020, TrueFi provided for (a) vetted borrowers to request loans denominated in TrueUSD ("TUSD"), (b) TRU Stakers to assess the creditworthiness of loans, (c) and TrueUSD lenders to earn attractive APY & TRU incentives on stablecoins loaned on the protocol.

Since that launch, TrueFi has evolved rapidly following a public roadmap, undergone two major protocol upgrades, started decentralizing protocol governance via Snapshot, and exceeded $200 million in loan originations with zero defaults — making TrueFi DeFi's first and leading uncollateralized lending protocol. This litepaper was updated July 2021 to include these milestones & reflect changes in the design of the protocol. While much of DeFi's success has been built on overcollateralized lending, uncollateralized lending and bringing true credit scoring to crypto is widely seen as the next transformative step for DeFi.

The traditional unsecured lending market makes up a $11 trillion global industry — yet none of that lending had come on-chain until TrueFi completed DeFi's first uncollateralized loan in 2020. Because uncollateralized lending provides an opportunity for lenders to earn higher long-term returns than secured lending, and for borrowers to maximize their capital efficiency, we believe on-chain, collateral-free lending will ultimately far outpace DeFi's existing collateralized lending market.

The project Ragnarok is a completely new version of the protocol, completely disconnected from the currently deployed contracts and kind of a Lending Marketplace.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts/proxy/utils/Initializable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts/token/ERC20/ERC20Upgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts/token/ERC20/IERC20Upgradeable.sol
@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/proxy/ERC1967/ERC1967Proxy.sol
@openzeppelin/contracts/proxy/utils/UUPSUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/proxy/utils/UUPSUpgradeable.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC721/ERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/token/ERC721/ERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/token/ERC721/IERC721Receiver.sol
@openzeppelin/contracts/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/utils/Address.sol

Dependency / Import Path	Source
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/utils/cryptography/ECDSA.sol
@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.4.0/contracts/utils/cryptography/draft-EIP712.sol

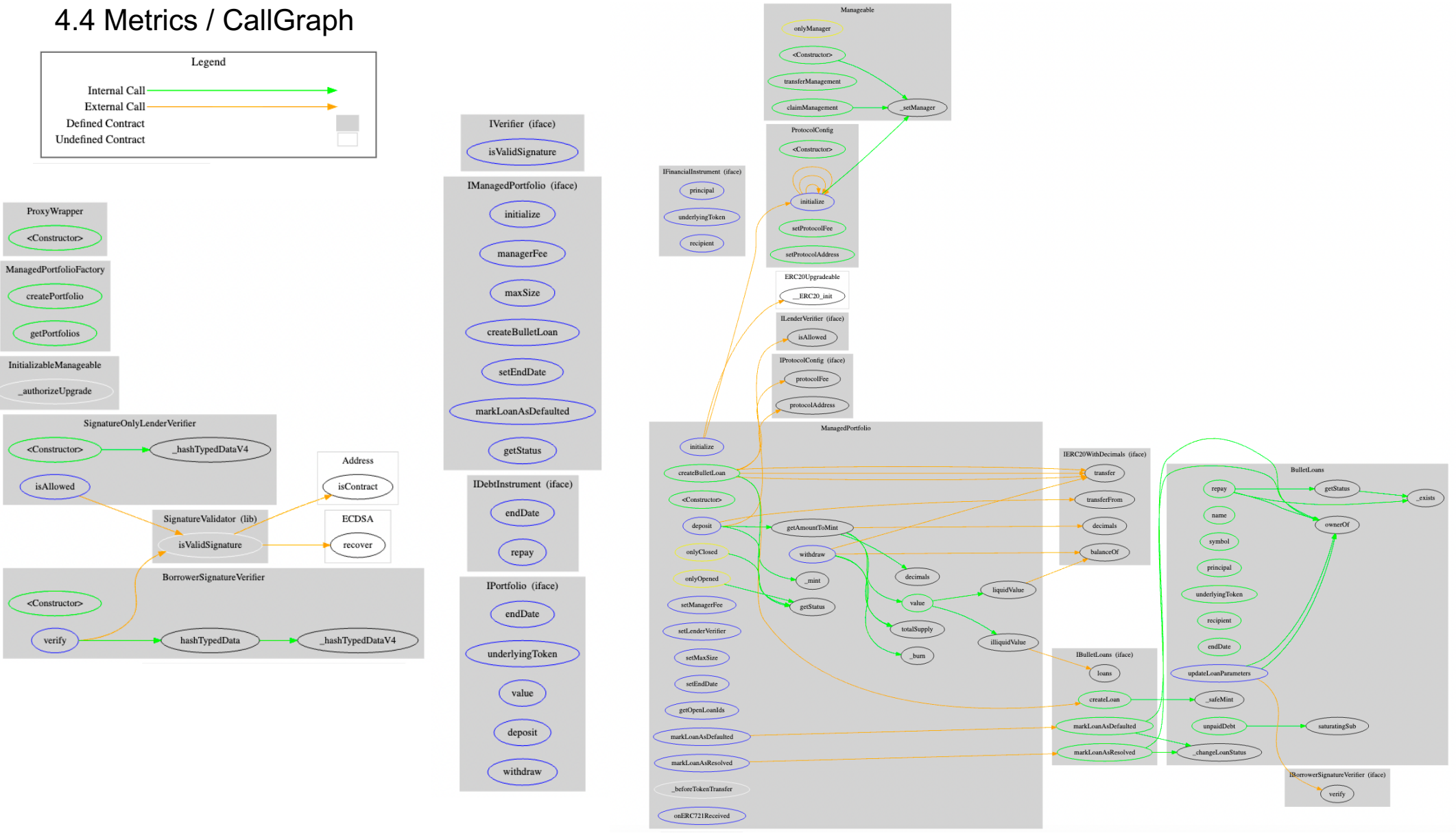
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

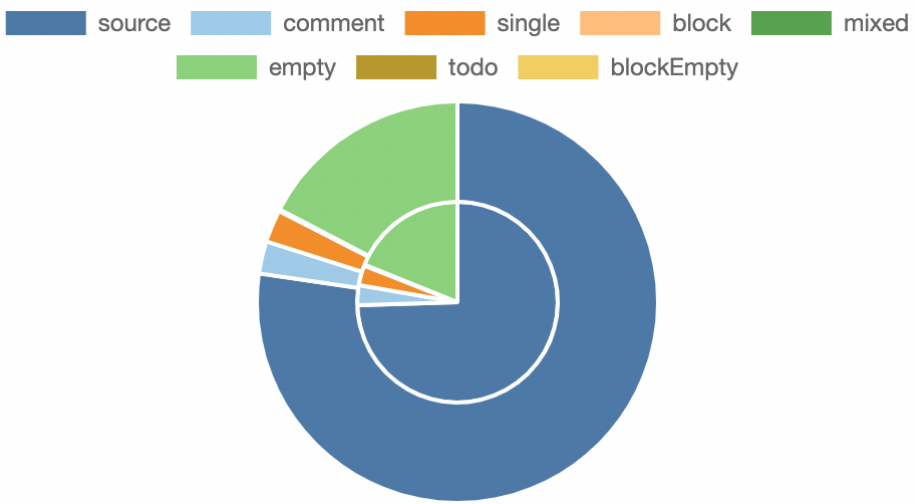
File	Fingerprint (MD5)
interfaces/IPortfolio.sol	2ac059e2df0dcc2f873c112d463b2a8e
interfaces/IDebtInstrument.sol	272b22e02f835d867b40499d52d8d188
interfaces/IManagedPortfolio.sol	46ab43d982cd4044801217380150254e
interfaces/IVerifier.sol	3728914240b07f948e5dfad136456261
interfaces/IBulletLoans.sol	57b8fec21939d437f055ebcfcf73177e
interfaces/ILenderVerifier.sol	67ba6446120a43f50aa6a0f127224a59
interfaces/IProtocolConfig.sol	f73909048b62a547eed4effaeec82ce6
interfaces/IFinancialInstrument.sol	ea2ff06bfaf4148cbf382f168ffb4fdd
interfaces/IERC20WithDecimals.sol	ce0d3ad3ebbd6cd7cac5c28ed45ebc71
interfaces/IBorrowerSignatureVerifier.sol	36d56f4db41ee5e4e76cac5ca78c0707
SignatureOnlyLenderVerifier.sol	d0f7225947a693b63faf552fddab0f40
libs/SignatureValidator.sol	e1de454e76d2a3f8018155096f1f3810
BorrowerSignatureVerifier.sol	b82a280dd8924ee8cb32bfcc5087624c
ManagedPortfolio.sol	35dee6675b7c3fba22903fdf3fbdd990

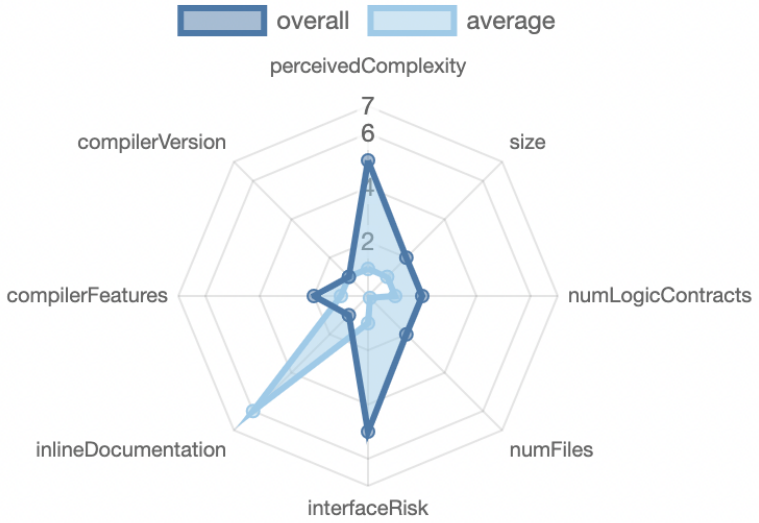
ProtocolConfig.sol	a569bdaab00f63af9c43ca40f64f46fa
access/Manageable.sol	8b07fabb879a672c63bc3a5df23492a2
access/InitializableManageable.sol	0600f064cf5c8f9a792846dbb83029ef
BulletLoans.sol	8281f51c599b0f607c33eaf79033297d
ManagedPortfolioFactory.sol	c96784b68c58de80796a28491973e839
proxy/ProxyWrapper.sol	5fb078ce7f142fa2dc9bcc7eb41a249d

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`0.8.10`		`yes`		

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪪 ECRecover	🌀 New/Create/Create2
`yes`			`yes`		`yes` → `NewContract:ProxyWrapper`

Exposed Functions
This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
70	1

External	Internal	Private	Pure	View
45	52	2	4	34

StateVariables

Total	🌐Public
32	22

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	interfaces/IPortfolio.sol		1	17	8	5	1	13	
🔍	interfaces/IDebtInstrument.sol		1	10	7	4	1	7	
🔍	interfaces/IManagedPortfolio.sol		1	46	17	13	1	17	
🔍	interfaces/IVerifier.sol		1	6	5	3	1	3	
🔍	interfaces/IBulletLoans.sol		1	40	15	11	1	11	
🔍	interfaces/ILenderVerifier.sol		1	10	5	3	1	3	
🔍	interfaces/IProtocolConfig.sol		1	8	5	3	1	5	
🔍	interfaces/IFinancialInstrument.sol		1	13	8	5	1	9	
🔍	interfaces/IERC20WithDecimals.sol		1	8	7	4	1	5	
🔍	interfaces/IBorrowerSignatureVerifier.sol		1	12	5	3	1	3	
📝	SignatureOnlyLenderVerifier.sol	1		28	24	18	1	21	
📚	libs/SignatureValidator.sol	1		23	19	15	2	11	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	BorrowerSignatureVerifier.sol	1		34	24	18	1	20	🧱
📝	ManagedPortfolio.sol	1		237	212	172	1	131	🔺
📝	ProtocolConfig.sol	1		31	31	23	1	20	
📝	access/Manageable.sol	1		34	34	26	1	14	
🎨	access/InitializableManageable.sol	1		17	17	12	1	11	
📝	BulletLoans.sol	1		171	156	123	1	79	
📝	ManagedPortfolioFactory.sol	1		63	51	43	1	32	🌀
📝	proxy/ProxyWrapper.sol	1		9	9	5	2	7	💰
📝📚🔍🎨	**Totals**	**10**	**10**	**817**	**659**	**509**	**22**	**422**	💰🔺🧱🌀🔆

Legend: [▔]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

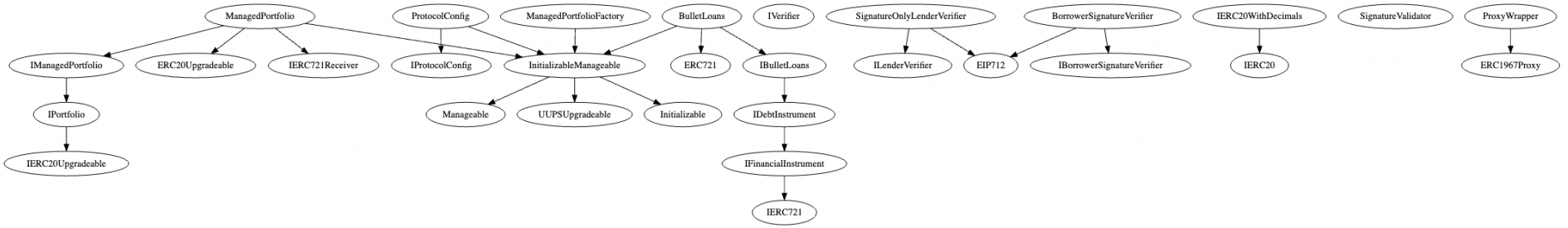
5. Scope of Work

The TrustToken Team provided us with the files that needs to be tested. The scope of the audit are the Ragnarok protocol contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way
- Changing the protocol address to 0x0 (onlyManager) is not leading to losing funds.
- Changing the protocol fee to 100% can not drain out user funds.
- Portfolio creation is setting name, symbol, duration, underlying token and manager fee and working as expected.
- Mathematical calculation inside the contracts is working fine and as expected.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

5.1.1 No return of overpaid dept
Severity: HIGH
Status: ACKNOWLEDGED
File(s) affected: BulletLoans.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation of the repay function in BulletLoans the user can overpay his dept and won't get back the overpaid amount. The fully entered repaid amount will be transferred to the lender.	Line: 66 BulletLoans.repay	We recommend to add a check, if the entered repayment amount is higher than the actual dept and transfer back the overpaid amount to the payee/borrower.

MEDIUM ISSUES

5.1.2 Missing access control
Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: BulletLoans.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation is no access control for the createLoan function in BulletLoans. Anyone could create a loan (mint BulletLoan token) without paying a principal to the borrower.	Line 41 BulletLoans.createLoan	Add access control to the createLoan function to ensure only managed portfolios are able to create new loans. By creating a new loan lend funds have to be transferred to the borrower.

5.1.3 Unintended use of defaulting loans
Severity: MEDIUM
Status: ACKNOWLEDGED
File(s) affected: BulletLoans.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation the lender can mark any loan at any time as defaulted even if the repayment date is not met.	Line: 79 BulletLoans.markLoanAsDefaulted	It is recommended to include a check if the repayment date is reached before changing the status of a loan to defaulted. This prevents defaulting loans before repayment date is met.

LOW ISSUES

5.1.4 Missing borrower allowance
Severity: LOW
Status: ACKNOWLEDGED
File(s) affected:

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation any lender can create a loan for any borrower. The borrower does not have to give a commitment to get a loan. If the loan is not paid back, the borrower may get problems for a loan he never requested.	NA	It is highly recommended to verify if a borrower really wants to get the specified loan. Therefore, a signature schema to validate a borrower's request with the given loan is required.

5.1.5 Missing natspec documentation
Severity: LOW
Status: ACKNOWLEDGED
Code: CWE-1056
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and	NA	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make

more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).		it easier to review and understand your smart contract. There are already in-line comments inside the codebase, but it can be increased.

5.1.6 Variable initialization with default value
Severity: LOW
Status: ACKNOWLEDGED
File(s) affected: ManagedPortfolio.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation are some variables explicitly initialized with their default value. These variables would have the same value implicitly.	ManagedPortfolio.illiquidValue
	uint256 _value = 0;
	uint256 i = 0;
		We recommend to remove the explicit initialization of default variable values to reduce gas consumption. By default defined variables are set to the default value implicitly and do not need to be reassigned.

INFORMATIONAL ISSUES

5.1.7 Unexplicit state variable visibility
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: BorrowerSignatureVerifier.sol, BulletLoans.sol, ManagedPortfolio.sol, SignatureOnlyLenderVerifier.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several state variables are leaking explicit visibility. Implicitly these variables are defined as public.	BorrowerSignatureVerifier.DOMAIN_NAME BorrowerSignatureVerifier.DOMAIN_VERSION BorrowerSignatureVerifier.NEW_LOAN_PARAMETERS_TYPEHASH BulletLoans.nextId ManagedPortfolio.YEAR ManagedPortfolio._loans SignatureOnlyLenderVerifier.DOMAIN_NAME SignatureOnlyLenderVerifier.DOMAIN_VERSION SignatureOnlyLenderVerifier.NEW_LOAN_PARAMETERS_TYPEHASH	Add explicit visibility types to all state variables to ensure availability and desired access control for the lowest possible gas costs. Ref.: https://docs.soliditylang.org/en/v0.8.11/contracts.html#visibility-and-getters

5.1.8 Inefficient storing of uints inside a struct
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: BulletLoans.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation uint variables inside the LoanMetadata struct are using two storage slots as the duration and repaymentDate are defined as uint256.	BulletLoans.LoanMetadata
	struct LoanMetadata {
	IERC20 underlyingToken;
	BulletLoanStatus status;
	uint256 principal;
	uint256 totalDebt;
	uint256 amountRepaid;
	uint256 duration;
	uint256 repaymentDate;
	address recipient;
	}
		It is recommended to change the uint256 variables to uint128 in order to use only one storage slot instead of two. The uint128 unit has more than enough space for time driven values such as time stamps. This leads to a lower gas consumption.
		
		Ref.:
		https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html

5.1.9 Unnecessary functions
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: BulletLoans.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation of BulletLoans are redundant functions defined. BulletLoan is	Line 91 / 95 function name() public pure override returns (string memory) {	We recommend to remove the name and symbol function from BulletLoans to decrease contract size and gas consumption by contract creation.

inheriting from ERC721 and is creating it with the desired name and symbol variable. ERC721 implements the functions name() and symbol() to return the given name and symbol. There is no need to override the functions in the BulletLoan source code.	return "BulletLoans";
	}
	
	function symbol() public pure override returns (string memory) {
	return "BulletLoans";
	}	

5.1.10 Unused function variables
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: ManagedPortfolio.sol, ILenderVerifier.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation some function definitions are holding variables which are not used inside the function. The withdraw function in ManagedPortfolio holds an unused parameter and the isAllowed function in the implementing contracts of the ILenderVerifier interface are not using the amount variable.	Line 93 ManagedPortfolio.withdraw function withdraw(uint256 sharesAmount, bytes memory) external onlyClosed returns Line 4 ILenderVerifier.isAllowed function isAllowed(address lender, uint256 amount, bytes memory signature) external view returns (bool);	We recommend to remove all variable/types which are not used inside the function to decrease gas consumption by calling this functions.

5.1.11 Redundant require checks
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: ManagedPortfolio.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation are several redundant require checks in multiple functions. The block time stamp is checked against the end date in the modifiers (status) as well as in additional require checks.	ManagedPortfolio.deposit onlyOpened modifier require(block.timestamp < endDate, "..."); ManagedPortfolio.createBulletLoan require(getStatus() != ManagedPortfolioStatus.Closed, "..."); require(block.timestamp < endDate, "...");	It is recommended to remove all redundant require checks to decrease code size and gas consumption.

5.1.12 Unneeded variable passed to initialize function
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: ManagedPortfolio.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation a**n** unneeded variable is passed to the ManagedPortfolio initialize function. The manager variable is always set to msg.sender.	Line 54 ManagedPortfolioFactory.createPortfolio Line 61 & Line 70 ManagedPortfolio.initialize	Thus the msg.sender does not change from createPortfolio function in PortfolioFactory to initialize function in ManagedPortfolio, it is recommended to remove the manager variable from the initialize function and use msg.sender instead.

5.1.13 Unused state variable
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
File(s) affected: ManagedPortfolioFactory.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation of PortfolioFactory a whitelist variable is defined but never used in any code.	ManagedPortfolioFactory.isWhitelisted	We recommend to use the whitelist to check access to the createPortfolio function as this is the only possible use case inside the contract. If this behavior is not intended it is recommended to remove the whitelist variable and the Managable functionality from the contract, because it would be unneeded and has no effects.

5.2 Verify claims

5.2.1 The smart contract is coded according to the newest standards and in a secure way
Status: tested and verified ✅
Description: Please check the open issues

5.2.2 Changing the protocol address to 0x0 (onlyManager) is not leading to losing funds.
Status: tested and verified ✅
Description: The protocol address receives a fee on every new loan creation (ManagedPortFolio.createBulletLoan line 125). If the protocol address is set to zero address and the underlying ERC20 token implementation does not prevent transferring funds to the zero address, the fee is lost. If the underlying ERC20 token implementation prevents transferring funds to the zero address, which is the standard behavior implemented by OpenZeppelin, the creation of new loan will fail with zero address as protocol address.

5.2.3 Changing the protocol fee to 100% cannot drain out user funds.
Status: tested and verified X
Description: The manager and protocol fees are calculated based on the principal amount and loan duration. The lender has to pay the fees on a new loan creation. These fees are transferred in addition to the principal amount from the users portfolio. The higher the protocol fee is, the more tokens are transferred to the protocol address in addition to the principal amount. The protocol fee can have any value even above 100%. The setter function in ProtocolConfig does not check for a maximum fee value. The users funds can be drained out the ManagedPortfolio by setting the protocol fee to a desired value.
Recommendation:
Implement a maximum value check for protocolFee in ProtocolConfig.
To avoid high extra costs for the lender it would be possible to let the borrower pay the protocol fees by substracting the fee from the principal amount. This would reduce the principal amount received by the borrower but will never transfer more than the principal amount from the lenders portfolio.

5.2.4 Portfolio creation is setting name, symbol, duration, underlying token and manager fee and working as expected.
Status: tested and verified ✅
Description: All variables are set as expected during portfolio creation. Thus, the manager variable does not need to be passed to the initialize function and msg.sender can be used instead.

5.2.5 Mathematical calculation inside the contracts is working fine and as expected.
Status: tested and verified ✅
Description: All mathematical operations are secure and working as expected.

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the January 20, 2022.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, no critical issues were found, after the manual and automated security testing and some claims have been successfully verified. Please check the open issues and our recommendations.

7. Deployed Smart Contract

PENDING