



**DeltaPrime**

**Protocol**

**SMART CONTRACT AUDIT**

**04.11.2022**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer.....	4
2. About the Project and Company .....	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level .....	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology .....	8
5. Metrics .....	9
5.1 Tested Contract Files .....	9
5.2 Used Code from other Frameworks/Smart Contracts .....	13
5.3 CallGraph.....	16
5.4 Inheritance Graph .....	17
5.5 Source Lines & Risk.....	18
5.6 Capabilities .....	19
5.7 Source Unites in Scope .....	21
6. Scope of Work.....	25
6.1 Findings Overview .....	27
6.2 Manual and Automated Vulnerability Test.....	28
6.2.1 The Owner Can Manipulate The Rate Calculation .....	28
6.2.2 Race Condition .....	29
6.2.3 All The Upgrades Should Be Performed Using A DAO Structure .....	30
6.2.4 Possible Front-Run In The contract initialization.....	31
6.2.5 Protocol Doesn't Count For Leap Years.....	32

6.2.6 Owner Can Renounce Ownership.....	33
6.2.7 Approve Race Condition .....	34
6.2.8 Diamond Facet Upgrades Should Be Performed In A Single Transaction .....	35
6.2.9 Missing Validation On amount .....	36
6.2.10 Possible DDoS In The Borrow .....	37
6.2.11 Missing Verification In Constructor For Duplicate Values .....	38
6.2.12 Floating Pragma Version Identified .....	39
6.2.13 Unnecessary Initializations .....	40
6.2.14 Function State Mutability Can Be Restricted To Pure.....	40
6.2.15 Remove Existing Declaration .....	42
6.2.16 Remove Unused Local Variables .....	42
6.3 SWC Attacks .....	43
6.4 Verify Claims .....	47
7. Executive Summary.....	48
8. About the Auditor .....	49

## 1. Disclaimer

The audit makes no statements or warranties 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 Prime Labs Distributed Ltd. 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 (21.09.2022)	Layout
0.4 (24.09.2022)	Automated Security Testing Manual Security Testing
0.5 (27.09.2022)	Verify Claims and Test Deployment
0.6 (31.09.2022)	Testing SWC Checks
0.9 (05.09.2022)	Summary and Recommendation
1.0 (07.10.2022)	Final document
1.1 (20.10.2022)	Re-check
1.2 (27.10.2022)	Re-check
1.3 (TBA)	Deployment to Mainnet

## 2. About the Project and Company

### **Company address:**

Prime Labs Distributed Ltd  
Intershore Chambers  
Geneva Place, 3rd floor  
ROAD TOWN, TORTOLA  
VG1110 British Virgin Islands



**Website:** <https://deltaprime.io>

**Twitter:** <https://twitter.com/DeltaPrimeDefi>

**Discord:** <https://discord.gg/57EdDsvhxK>

**GitHub:** <https://github.com/DeltaPrimeLabs>

**Medium:** [https://medium.com/@Delta\\_Prime](https://medium.com/@Delta_Prime)

**Documentation:** <https://docs.deltaprime.io>

## 2.1 Project Overview

DeltaPrime is a decentralised borrowing and investing ecosystem, unlocking trapped liquidity by increasing capital efficiency. On DeltaPrime users can easily deposit and borrow funds to increase the power of their usual DeFi investments. The minimum collateral ratio for these loans is 20%.

Borrowed funds can be used to invest in integrated protocols. As DeltaPrime does not discriminate against protocols, we aim to integrate the vast majority of popular protocols in DeFi, provided that they are considered secure. Right now that assessment is made by the development team. In the near future DeltaPrime will become a Decentralised Autonomous Organisation (DAO), giving you full control over protocol integrations.

### Why DeltaPrime?

At the time of writing 50 billion dollars worth of assets are locked in lending protocols. The vast majority of these funds are locked as collateral, providing little to no value to the crypto community. DeltaPrime returns these funds back to the users. This potential multi-billion dollar liquidity unlocking benefits borrowers, depositors and integrated protocols alike.

When you borrow and invest on DeltaPrime you get:

- Extra capital to increase your returns
- No credit checks
- The best DeFi protocols in one Account

When you deposit on DeltaPrime:

- Your deposit improves the health of the blockchain
- Your deposit is protected by safety features in the smart contracts
- You can expect a higher interest rate due to increased capital-efficiency

### 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.



## 5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

### 5.1 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)
./contracts/VariableUtilisationRatesCalculator.sol	d8b1b2989578de332bc7aa03c5223f58
./contracts/abstract/ECDSAVerify.sol	fca5485a69c157b8ad2ed4a9c46d9439
./contracts/SmartLoanDiamondBeacon.sol	fe98de0ad0864b4950cf0a18585f03a5
./contracts/facets/celo/UbeswapDEXFacet.sol	209d83db49b3ffa77f80e07b188d4218
./contracts/facets/DiamondCutFacet.sol	c32ee6d1a7964fada2fdbe5c59917e7a
./contracts/facets/OwnershipFacet.sol	51c10fc7c06aa49e292d5c0c75af38fd
./contracts/facets/SmartLoanViewFacet.sol	9728290a7ef61bb7391c8353e691c51e
./contracts/facets/DiamondInit.sol	ac88bd35d773f9f1100be97d8547b9d4
./contracts/facets/UniswapV2DEXFacet.sol	8541d3b587177b95abbb6f7789d1a30f
./contracts/facets/DiamondLoupeFacet.sol	896f18907f10fc90b699712dc5da6b04
./contracts/facets/AssetsOperationsFacet.sol	286619a0c06f9f3a93581289c852fd32
./contracts/facets/SmartLoanWrappedNativeTokenFacet.sol	d0b512655a658c617c2b5838119a9b44
./contracts/facets/avalanche/TraderJoeDEXFacet.sol	1d90e8c1f7c302ab338ce7ff95fe0493
./contracts/facets/avalanche/VectorFinanceFacet.sol	6c98a6e54be8fb23e144bfd4114d4b3d
./contracts/facets/avalanche/YieldYakFacet.sol	119c1d7da96244c0eae1db97621c20ec
./contracts/facets/avalanche/PangolinDEXFacet.sol	4c03e68fcbbe998a93957826d5319f7

./contracts/facets/SmartLoanLiquidationFacet.sol	f463301e984f2439bf3268cf33265a0c
./contracts/facets/SolvencyFacet.sol	6d809eee081755f8bf07d26d6822ba4e
./contracts/CompoundingIndex.sol	9a65ebc6b507ca97b68fbcb48ae8f11
./contracts/Pool.sol	2b00f5663e9ca0f020af65d9148f0b47
./contracts/TokenManager.sol	4c2274bfdfb31c9a6259ce2e0432cc61
./contracts/DiamondHelper.sol	8fa93a24aad2e4f40eb2b07536c36d27
./contracts/WrappedNativeTokenPool.sol	ada13316b615edd4224c4e1d2ea37bcd
./contracts/TokenList.sol	36051199ffb3a03f12b6f61122e83e7e
./contracts/integrations/celo/UbeswapIntermediary.sol	9a40273b1ff9a2f34a73364f4f4cb782
./contracts/integrations/UniswapV2Intermediary.sol	a0f43cc4e80ce8aad74c4db0d7ad6d13
./contracts/integrations/avalanche/TraderJoelIntermediary.sol	39d2e6c987bd15be202a928380cf4f84
./contracts/integrations/avalanche/PangolinIntermediary.sol	56c9505dca041a6f734b5a90701c8c95
./contracts/LinearIndex.sol	e9f50d5e23f107958bed5157ed6a48f5
./contracts/ReentrancyGuardKeccak.sol	68262ae4519fed12e9add0390efb327c
./contracts/OnlyOwnerOrInsolvent.sol	f29ed1eaf79ac181ed3392a5374afdc2
./contracts/RedstoneConfigManager.sol	f71dc51b80f8e66136c5a37cf8096a7d
./contracts/lib/celo/DeploymentConstants.sol	e3f40e255dcffbd65b01a2c4df544c10
./contracts/lib/SolvencyMethods.sol	e43141619805a8fff302fbc0c7ac251e
./contracts/lib/DiamondStorageLib.sol	6370e3db18a605088085402289ad8d08
./contracts/SmartLoansFactory.sol	7f179bb47b1827b859ed51ee31f6ec7f
./contracts/proxies/openzeppelinVirtual/BeaconProxyVirtual.sol	f728e98201bf1167c4425556712720eb
./contracts/proxies/openzeppelinVirtual/ERC1967UpgradeVirtual.sol	b4a9eedd1ab0adc8ce866b351c61a6d3
./contracts/proxies/SmartLoanDiamondProxy.sol	18d2693a03ac8e38c43bff09742ac1bd
./contracts/interfaces/IAssetsExchange.sol	dfb31d32fe7650cd9e9bd95738b6e173
./contracts/interfaces/IIndex.sol	eb1d1f4d8e1b4df7b63829098df628cf
./contracts/interfaces/facets/celo/IUbeswapDEXFacet.sol	c76221c8a8f937fafeb463e0e29ffab5

./contracts/interfaces/facets/IAssetsOperationsFacet.sol	9ca10e96b6c616682dfa34f493a0b8c1
./contracts/interfaces/facets/IOwnershipFacet.sol	9c99c6f1a4fa6d76e16e239bcfce2241
./contracts/interfaces/facets/IDiamondInit.sol	134beacca4492c2643a94880152217f8
./contracts/interfaces/facets/ISolvencyFacet.sol	b3317736a365dcfb6bd6125cdf892716
./contracts/interfaces/facets/ISmartLoanLiquidationFacet.sol	8946893a7b1085d0f2f485968a295f5a
./contracts/interfaces/facets/avalanche/IYieldYakFacet.sol	d455edb00cdffa1fdb255fe6d4386591
./contracts/interfaces/facets/avalanche/ITraderJoeDEXFacet.sol	2293963e8b6e3078063932c00ba41a20
./contracts/interfaces/facets/avalanche/IVectorFinanceFacet.sol	3fda13d767279b818b32b2bd63b10f66
./contracts/interfaces/facets/avalanche/IPangolinDEXFacet.sol	9bc12f40124fd3a607347ccea8580cd3
./contracts/interfaces/facets/avalanche/IYakStakingVectorSAV2.sol	5d756adb68fd7f785552988a196437b9
./contracts/interfaces/facets/avalanche/IYakStakingAVAXAAVEV1.sol	ac78140c10a59b0cf36ec738b36195c9
./contracts/interfaces/facets/IUniswapV2DEXFacet.sol	07037be1ba2721eb22c9513eb7e7766f
./contracts/interfaces/facets/ISmartLoanWrappedNativeTokenFacet.sol	e9e7ef4fc9eec239c7ef23a45f5d3076
./contracts/interfaces/facets/ISmartLoanViewFacet.sol	c61426b0eb4d0c2e512b57c52c85ff70
./contracts/interfaces/IDiamondCut.sol	270717b86dd309fd332df77ae0595f30
./contracts/interfaces/IRatesCalculator.sol	438b76627d9d575beea9340143ecabd8
./contracts/interfaces/IVectorFinanceStaking.sol	28df0618d4e690d4e02bc575934219e2
./contracts/interfaces/IWrappedNativeToken.sol	ef03daa42d3e1d20371361e1dd968708
./contracts/interfaces/IStakingPositions.sol	d52d3de2c8e954e1e453cf13fb997844
./contracts/interfaces/IBorrowersRegistry.sol	c42f5c22fb395e24438afe8bdc2e6561
./contracts/interfaces/SmartLoanGigaChadInterface.sol	aec5cdfc139e51399ce13a2abb992bc8
./contracts/interfaces/IDiamondLoupe.sol	0c085e9e45b787941bff107d40cd76b1

./contracts/interfaces/IERC173.sol	98c90d910ce93004f6d1c83dc8480305
./contracts/interfaces/IDiamondBeacon.sol	9c9d8371c32b4021a16acf9b667b58aa

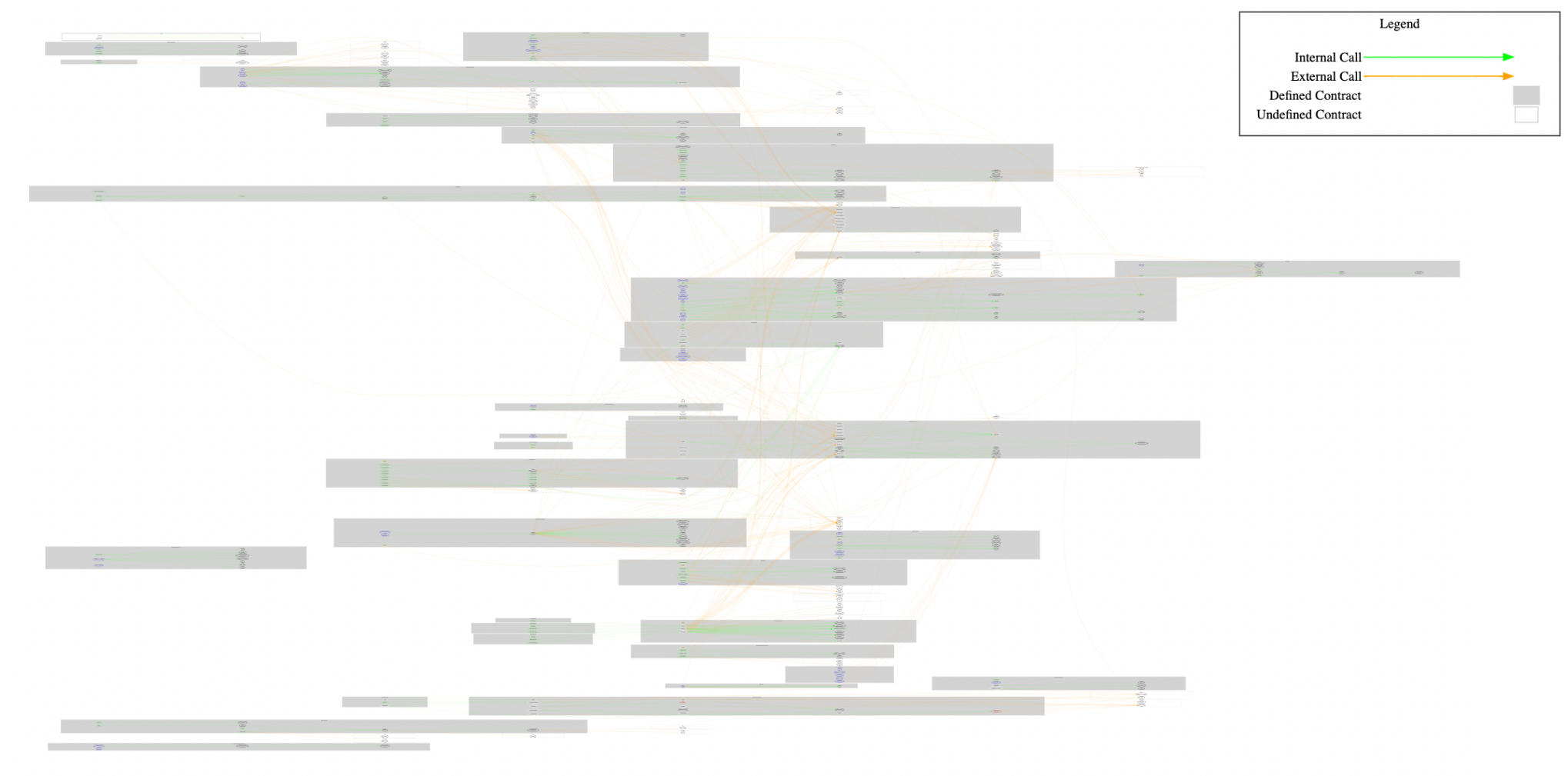
## 5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts</a>
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts</a>
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.1/contracts</a>
@openzeppelin/contracts/access/Ownable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts</a>
@openzeppelin/contracts/interfaces/draft-IERC1822.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts</a>
@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts</a>
@openzeppelin/contracts/proxy/Proxy.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts</a>
@openzeppelin/contracts/proxy/beacon/BeaconProxy.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts</a>
@openzeppelin/contracts/security/ReentrancyGuard.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts</a>

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/IERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/token/ERC20/IERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/token/ERC20/IERC20.sol</a>
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/token/ERC20/extensions/IERC20Metadata.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/token/ERC20/extensions/IERC20Metadata.sol</a>
@openzeppelin/contracts/utils/Address.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/Address.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/Address.sol</a>
@openzeppelin/contracts/utils/StorageSlot.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/StorageSlot.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/StorageSlot.sol</a>
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/cryptography/ECDSA.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/cryptography/ECDSA.sol</a>
@openzeppelin/contracts/utils/introspection/IERC165.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/introspection/IERC165.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/introspection/IERC165.sol</a>
@openzeppelin/contracts/utils/math/Math.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/math/Math.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/utils/math/Math.sol</a>
@uniswap/lib/contracts/libraries/TransferHelper.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/libraries/TransferHelper.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.1.0/contracts/libraries/TransferHelper.sol</a>
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	<a href="https://github.com/Uniswap/v2-core/tree/master/contracts/interfaces/IUniswapV2Factory.sol">https://github.com/Uniswap/v2-core/tree/master/contracts/interfaces/IUniswapV2Factory.sol</a>
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	<a href="https://github.com/Uniswap/v2-periphery/tree/master/contracts/interfaces/IUniswapV2Router01.sol">https://github.com/Uniswap/v2-periphery/tree/master/contracts/interfaces/IUniswapV2Router01.sol</a>

Dependency / Import Path	Source
redstone-evm-connector/lib/contracts/commons/ProxyConnector.sol	<a href="https://github.com/redstone-finance/redstone-evm-connector/tree/master/contracts/commons/ProxyConnector.sol">https://github.com/redstone-finance/redstone-evm-connector/tree/master/contracts/commons/ProxyConnector.sol</a>
redstone-evm-connector/lib/contracts/message-based/PriceAware.sol	<a href="https://github.com/redstone-finance/redstone-evm-connector/tree/master/contracts/message-based/PriceAware.sol">https://github.com/redstone-finance/redstone-evm-connector/tree/master/contracts/message-based/PriceAware.sol</a>

## 5.3 CallGraph



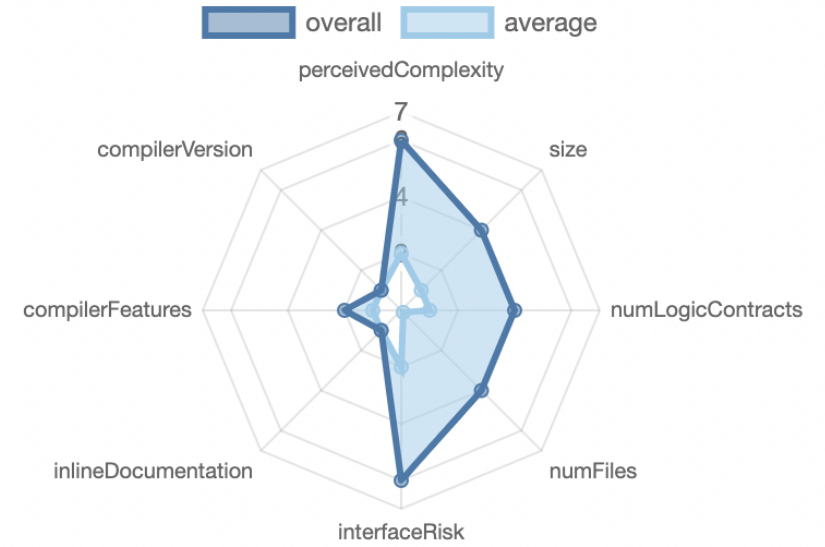


## 5.4 Inheritance Graph



## 5.5 Source Lines & Risk

source comment single block mixed  
empty todo blockEmpty





## 5.6 Capabilities


Solidity Versions observed		🔧 Experimental Features	💰 Can Receive Funds	💻 Uses Assembly	💣 Has Destroyable Contracts
<code>^0.8.17</code> <code>&gt;=0.7.0 &lt;0.9.0</code> <code>^0.8.0</code> <code>^0.8.4</code> <code>^0.8.2</code>			yes	yes (7 asm blocks)	
📡 Transfers ETH	⚡ Low-Level Calls	👤 DelegateCall	🎲 Uses Hash Functions	📄 ECTRecover	🌀 New/Create/Create2
		yes	yes		yes → <code>NewContract:SmartLoanDiamondProxy</code>
♻️ TryCatch	Σ Unchecked				
yes	yes				
📄 Contracts	📖 Libraries	🔍 Interfaces	🎨 Abstract		
33	2	0	4		

## Exposed Functions

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

 <b>Public</b>	 <b>Payable</b>				
155	14				
<b>External</b>	<b>Internal</b>	<b>Private</b>	<b>Pure</b>	<b>View</b>	
48	221	4	11	94	












## StateVariables




















<b>Total</b>	 <b>Public</b>
88	31
















## 5.7 Source Unites in Scope







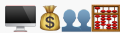

Source: <https://github.com/DeltaPrimeLabs/deltaprime-primeloans/tree/audit-v2-changes>

Last commit: 0e815117dbde8a104dbf6b0f59af5a870df26400

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/VariableUtilisationRatesCalculator.sol	1	_____	108	108	45	46	34	_____
	contracts/CompoundingIndex.sol	1	_____	109	109	50	38	46	_____
	contracts/WrappedNativeTokenPool.sol	1	_____	56	56	25	18	35	💰
	contracts/DiamondHelper.sol	1	_____	18	18	8	7	5	_____
	contracts/TokenManager.sol	1	_____	232	232	141	53	113	_____
	contracts/Pool.sol	1	_____	419	419	213	109	222	Σ
	contracts/SmartLoanDiamondBeacon.sol	1	_____	74	74	45	21	79	💻💰👥
	contracts/TokenList.sol	1	_____	98	98	62	15	51	_____
	contracts/LinearIndex.sol	1	_____	112	112	51	39	38	_____
	contracts/ReentrancyGuardKeccak.sol	1	_____	47	47	17	21	6	_____
	contracts/abstract/ECDSAVerify.sol	1	_____	16	16	10	2	7	🏠

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/facets/SmartLoanViewFacet.sol	1	_____	126	126	86	15	102	_____
	contracts/integrations/UniswapV2Intermediary.sol	1	_____	183	183	97	42	134	
	contracts/facets/OwnershipFacet.sol	1	_____	35	35	25	3	27	_____
	contracts/lib/DiamondStorageLib.sol	1	_____	299	287	219	32	138	
	contracts/facets/UniswapV2DEXFacet.sol	1	_____	193	193	115	39	135	
	contracts/facets/DiamondInit.sol	1	_____	52	52	19	22	11	_____
	contracts/integrations/avalanche/PangolinIntermediary.sol	1	_____	18	18	8	7	4	_____
	contracts/facets/SmartLoanLiquidationFacet.sol	1	_____	254	254	123	80	113	
	contracts/facets/ceelo/UbeswapDEXFacet.sol	1	_____	27	27	10	13	9	_____
	contracts/facets/SmartLoanWrappedNativeTokenFacet.sol	1	_____	70	70	30	23	35	
	contracts/integrations/avalanche/TraderJoelIntermediary.sol	1	_____	18	18	8	7	4	_____
	contracts/facets/AssetsOperationsFacet.sol	1	_____	163	163	68	65	83	
	contracts/facets/DiamondLoupeFacet.sol	1	_____	68	68	32	28	36	_____

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/lib/SolvencyMethods.sol	1	_____	105	105	76	18	42	_____
	contracts/facets/DiamondCutFacet.sol	1	_____	31	27	9	14	7	_____
	contracts/facets/SolvencyFacet.sol	1	_____	167	167	94	38	101	_____
	contracts/proxies/SmartLoanDiamondProxy.sol	1	_____	43	36	13	17	21	
	contracts/lib/ceio/DeploymentConstants.sol	1	_____	73	73	38	15	12	_____
	contracts/integrations/ceio/UbeswapIntermediary.sol	1	_____	31	31	15	11	15	_____
	contracts/SmartLoansFactory.sol	1	_____	131	131	78	26	91	
	contracts/facets/avalanche/TraderJoeDEXFacet.sol	1	_____	38	38	19	13	17	_____
	contracts/RedstoneConfigManager.sol	1	_____	61	61	44	7	46	_____
	contracts/OnlyOwnerOrInsolvent.sol	1	_____	30	30	15	9	9	_____
	contracts/facets/avalanche/PangolinDEXFacet.sol	1	_____	38	38	19	13	17	_____
	contracts/facets/avalanche/YieldYakFacet.sol	1	_____	178	178	83	52	116	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/facets/avalanche/VectorFinanceFacet.sol	1	_____	184	182	110	30	100	_____
	contracts/proxies/openzeppelinVirtual/ERC1967UpgradeVirtual.sol	1	_____	182	160	68	78	43	
	contracts/proxies/openzeppelinVirtual/BeaconProxyVirtual.sol	1	_____	45	45	17	23	17	
	<b>Totals</b>	<b>39</b>	_____	<b>4132</b>	<b>4085</b>	<b>2205</b>	<b>1109</b>	<b>2121</b>	 

#### 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, ...)



## 6. Scope of Work

The DeltaPrime team provided us with the files that needs to be tested. The scope of the audit are the protocol contracts.

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

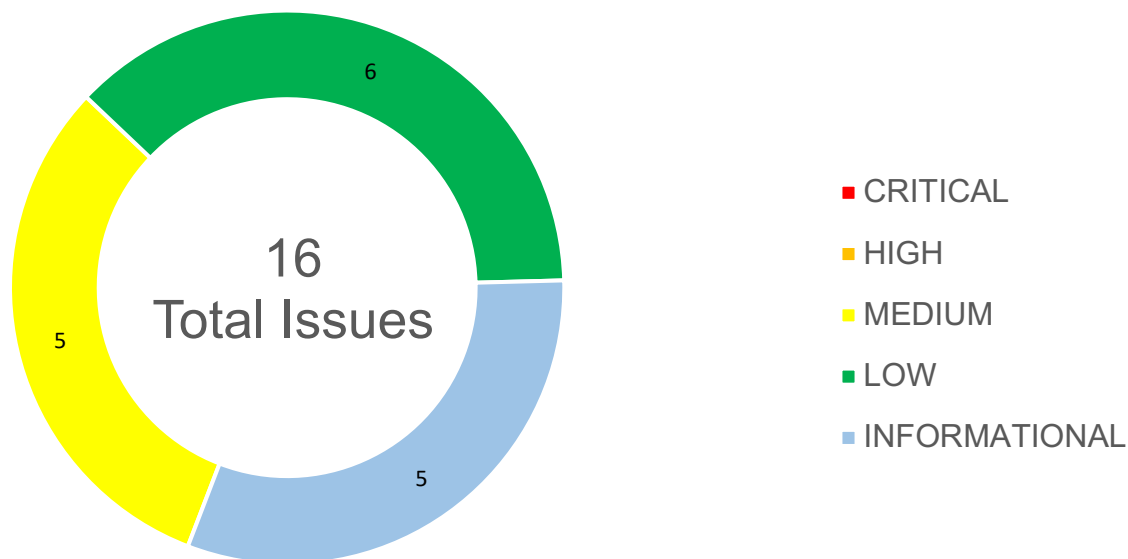
### Protocol related threats:

- Exploit through ERC20 methods
- Solvency - every method in a SmartLoan that could decrease its solvency has to check if it does not breach the solvency limit
- Oracle (PriceAware contracts) - important to check if data feeds could be manipulated by a user or third-party (contracts check signers of data packages)
- Upgradability - there are many places where proxy contracts are used. It's important to check if only admin wallets are able to upgrade the contracts and if they are correctly set.
- Wrong accounting of deposited/borrowed funds, accumulating deposits etc.
- Unauthorized party updating the list of whitelisted assets
- Unauthorized party accessing method when solvent
- Wrong handling of owner
- Wrong calculation of rates, especially near inflection points or in extreme pool utilizations like 0 or  $\geq 100\%$
- Unauthorized party changing configuration
- Wrong accounting of owners/loans lists
- Exploiting the admin ownership transfer proposal / execution
- Taking over the admin role / executing contractOwner-only methods (found in DiamondCutFacet.sol contract)
- Unauthorized funcSig=>facetAddress mapping edition
- **In general the biggest possible threat** would be if a bad actor would be able to (in any way) manipulate this contract to return an address of a malicious contract in a response to a **SmartLoanDiamondProxy (SmartLoan)** contract asking for an address to .delegatecall() to as this would mean execution of an arbitrary code in the context of user SmartLoan that holds custody over both his collateral as well as the borrowed funds.
- Wrong compounding of values
- Unauthorized party changing ownership
- Liquidation of a solvent account

- Any manipulation by a liquidator to exploit liquidated account
- Wrong evaluation of assets or debts
- Wrong staking/unstaking
- Wrong handling of reward assets
- Wrong handling of slippage
- Unauthorized addition/removal/update of methods' selectors=>addresses
- Unauthorized addition/removal of loan-owned assets

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.

## 6.1 Findings Overview



No	Title	Severity	Status
6.2.1	The Owner Can Manipulate The Rate Calculation	MEDIUM	ACKNOWLEDGED
6.2.2	Race Condition	MEDIUM	ACKNOWLEDGED
6.2.3	All The Upgrades Should Be Performed Using A DAO Structure	MEDIUM	ACKNOWLEDGED
6.2.4	Possible Front-Run In The contract initialization	MEDIUM	ACKNOWLEDGED
6.2.5	Protocol Doesn't Count For Leap Years	MEDIUM	ACKNOWLEDGED
6.2.6	Owner Can Renounce Ownership	LOW	FIXED
6.2.7	Approve Race Condition	LOW	ACKNOWLEDGED
6.2.8	Diamond Facet Upgrades Should Be Performed In A Single Transaction	LOW	FIXED
6.2.9	Missing Validation On amount	LOW	FIXED

6.2.10	Possible DDoS In The Borrow	LOW	ACKNOWLEDGED
6.2.11	Missing Verification In Constructor For Duplicate Values	LOW	FIXED
6.2.12	Floating Pragma Version Identified	INFORMATIONAL	FIXED
6.2.13	Unnecessary Initializations	INFORMATIONAL	ACKNOWLEDGED
6.2.14	Function State Mutability Can Be Restricted To Pure	INFORMATIONAL	FIXED
6.2.15	Remove Existing Declaration	INFORMATIONAL	FIXED
6.2.16	Remove Unused Local Variables	INFORMATIONAL	FIXED

## 6.2 Manual and Automated Vulnerability Test

### CRITICAL ISSUES

During the audit, Chainsulting's experts found **0 Critical issues** in the code of the smart contract.

### HIGH ISSUES

During the audit, Chainsulting's experts found **0 High issues** in the code of the smart contract.

### MEDIUM ISSUES

During the audit, Chainsulting's experts found **5 Medium issues** in the code of the smart contract.

#### 6.2.1 The Owner Can Manipulate The Rate Calculation

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Pool.sol

Update: The Gnosis Safe will indeed be used to mitigate the centralization risk. Later on this may be further extended to a DAO voting mechanism.

<b>Attack / Description</b>	Using the setRatesCalculator function, the owner can change the ratesCalculator that is responsible for changing the deposit and the borrowing rates, which allows the admin to manipulate the rate calculation by injecting the address of any contract. This represents a centralization risk where the owner has too much control over the pool's parameters.
<b>Code</b>	Line 64 - 73 (Pool.sol) <pre> function setRatesCalculator(IRatesCalculator ratesCalculator_) external onlyOwner {     // setting address(0) ratesCalculator_ freezes the pool     require(AddressUpgradeable.isContract(address(ratesCalculator_))    address(ratesCalculator_) == address(0), "Must be a contract");     ratesCalculator = ratesCalculator_;     if (address(ratesCalculator_) != address(0)) {         _updateRates();     }      emit RatesCalculatorChanged(address(ratesCalculator_), block.timestamp); } </pre>
<b>Result/Recommendation</b>	Consider verifying the integrity of the ratesCalculator_ address provided by the owner, this can be achieved by storing bytecode hash of the ratesCalculator contract as a constant, and comparing it with the bytecode hash of the contract provided in the arguments. If the business logic supports changing the rate's calculation, then a Multisignature wallet (Gnosis Safe) should be used to mitigate centralization risks.

### 6.2.2 Race Condition

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: VariableUtilisationRatesCalculator.sol



Update: Both the Diamond upgrading mechanism and well as all crucial setters will be only accessible through a publicly-known timelock contract.

<b>Attack / Description</b>	The setSpread function allows the owner to change the spread variable. If a user uses the rates calculated by the contract, and then the owner changes the spread variable, it is possible that the owner's transaction will get mined prior, which will generate an unexpected result for the user.
<b>Code</b>	Line 104 - 107 (VariableUtilisationRatesCalculator.sol)  <pre>function setSpread(uint256 _spread) external onlyOwner {     require(_spread &lt; 1e18, "Spread must be smaller than 1e18");     spread = _spread; }</pre>
<b>Result/Recommendation</b>	Consider documenting (event) this behaviour and notifying the community with any change on the spread value.

### 6.2.3 All The Upgrades Should Be Performed Using A DAO Structure

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: DiamondCutFacet.sol

Update: The Gnosis Safe will indeed be used to mitigate the centralization risk. Later on this may be further extended to a DAO voting mechanism. Both the Diamond upgrading mechanism and well as all crucial setters will be only accessible through a publicly-known timelock contract.

<b>Attack / Description</b>	The project makes use of the diamond pattern, this implementation offers the upgradability of functionalities along with other advantages. However, using this approach the contract owner can upgrade the contract to implement any logic, this represents a centralization risk. Means the owner can change the logic of the code anytime which can cause unexpected behaviours to the users.
-----------------------------	---

<b>Code</b>	Line 23 - 30 (DiamondCutFacet.sol) <pre> function diamondCut(     FacetCut[] calldata _diamondCut,     address _init,     bytes calldata _calldata ) external override {     DiamondStorageLib.enforceIsContractOwner();     DiamondStorageLib.diamondCut(_diamondCut, _init, _calldata); } </pre>
<b>Result/Recommendation</b>	Consider using a DAO as the contract owner to include the community in the decision of upgrades.

#### 6.2.4 Possible Front-Run In The contract initialization

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: SmartLoansFactory.sol, LinearIndex.sol

Update: Even though it's technically possible, we believe that the probability of this happening is low enough not to make any code adjustments. If the scenario was to happen - then we can always just like you mentioned deploy this contract and initialize it in a single tx.

<b>Attack / Description</b>	The initialization of SmartLoansFactory and LinearIndex contract requires calling the initialize function, However, there is no access control in place to avoid an attacker to front-run this call. As a result of that, an attacker could temporarily block the initialization and front-running each deployment, using invalid parameters in the initialize call.
<b>Code</b>	Line 56 - 59 (SmartLoansFactory.sol) <pre> function initialize(address payable _smartLoanDiamond) external initializer {     smartLoanDiamond = SmartLoanDiamondBeacon(_smartLoanDiamond); } </pre>

	<pre>         __Ownable_init();     }  Line 27 - 35 (LinearIndex.sol) function initialize(address owner_) external initializer {     index = BASE_RATE;     indexUpdateTime = block.timestamp;      __Ownable_init();     if (address(owner_) != address(0)) {         transferOwnership(owner_);     } } </pre>
<b>Result/Recommendation</b>	This issue can be solved by deploying the contract and calling the initialize function in one transaction or add some access control to the initialize function.

### 6.2.5 Protocol Doesn't Count For Leap Years

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: CompoundingIndex.sol, LinearIndex.sol

Update: We are aware of this, however this check would add additional gas usage which we believe is obsolete anyway as it is already acceptable in TradFi financial products agreements that the interest is calculated using 365 days even during leap years.

<b>Attack / Description</b>	<p>The getCompoundedFactor function calculates the compounded factor in Ray using the following formula : <math>((\text{rate.wadToRay()} / \text{SECONDS\_IN\_YEAR}) + \text{WadRayMath.ray}()).\text{rayPow}(\text{period})</math>; The SECONDS_IN_YEAR is equal to 365, and therefore the formula is incorrect if it's a leap year.</p> <p>The getLinearFactor function calculates the linear factor in Ray using the following formula: <math>\text{rate} * \text{period} * 1\text{e}9 / \text{SECONDS\_IN\_YEAR} + 1\text{e}27</math>; The SECONDS_IN_YEAR is equal to 365, and therefore the formula is incorrect if it's a leap year.</p>
-----------------------------	---



<b>Code</b>	<p>Line 99 - 101 (CompoundingIndex.sol)</p> <pre>function getCompoundedFactor(uint256 period) internal view returns (uint256) {     return ((rate.wadToRay() / SECONDS_IN_YEAR) + WadRayMath.ray()).rayPow(period); }</pre> <p>Line 102 - 104 (LinearIndex.sol)</p> <pre>function getLinearFactor(uint256 period) virtual internal view returns (uint256) {     return rate * period * 1e9 / SECONDS_IN_YEAR + 1e27; }</pre>
<b>Result/Recommendation</b>	Consider counting leap years in the protocol.

## LOW ISSUES

During the audit, Chainsulting's experts found **6 Low issues** in the code of the smart contract.

### 6.2.6 Owner Can Renounce Ownership

Severity: LOW

Status: **FIXED**

Code: NA

File(s) affected: Pool.sol, VariableUtilisationRatesCalculator.sol, SmartLoansFactory.sol, RedstoneConfigManager.sol

Update: That's a fair point - we don't need the renounceOwnership function, we only need the ability to transfer it.

Removed in commit [#2d52ce7fae2b5ae7273cd41d52909913ac5c745f](#)

<b>Attack / Description</b>	Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an owner, rendering some owner-exclusive functionality unavailable.
<b>Code</b>	<p>Line 21 (Pool.sol)</p> <pre>contract Pool is OwnableUpgradeable, ReentrancyGuardUpgradeable, IERC20 {</pre>

	<p>Line 16 (VariableUtilisationRatesCalculator.sol)</p> <pre>contract VariableUtilisationRatesCalculator is IRatesCalculator, Ownable {</pre> <p>Line 24 (SmartLoansFactory.sol)</p> <pre>contract SmartLoansFactory is OwnableUpgradeable, IBorrowersRegistry {</pre> <p>Line 7 (RedstoneConfigManager.sol)</p> <pre>contract RedstoneConfigManager is Ownable {</pre>
<b>Result/Recommendation</b>	<p>We recommend to prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable renounce ownership functionality by overriding it.</p>

### 6.2.7 Approve Race Condition

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Pool.sol

Update: We're leaving the approve() function for the sake of staying compliant with ERC-20 standard, however we do not use it anywhere in our contracts.

<b>Attack / Description</b>	<p>The standard ERC20 implementation contains a widely known racing condition in its approve function, wherein a spender can witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.</p>
<b>Code</b>	<p>Line 135 - 142 (Pool.sol)</p> <pre>function approve(address spender, uint256 amount) external override returns (bool) {</pre>

	<pre> require(spender != address(0), "Allowance spender cannot be a zero address"); _allowed[msg.sender][spender] = amount;  emit Approval(msg.sender, spender, amount);  return true; } </pre>
<b>Result/Recommendation</b>	We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

### 6.2.8 Diamond Facet Upgrades Should Be Performed In A Single Transaction

Severity: LOW

Status: **FIXED**

Code: NA

File(s) affected: DiamondCutFacet.sol

Update: Technically it's already not an issue as our current codebase supports upgrades affecting multiple facets in a single tx but we like the idea of pausing the fallback function for the duration of an upgrade.

Added pausability in commit [#d2d93a6f7c7d668d1e317597bcad6e4750ba63be](#)

<b>Attack / Description</b>	If during an upgrade diamondCut() calls are executed in multiple Ethereum transactions, users may be exposed to contracts that are upgraded only partially, i.e., some of the functions are upgraded while others are not. This may result in unexpected inconsistencies.
<b>Code</b>	<p>Line 23 - 30 (DiamondCutFacet.sol)</p> <pre> function diamondCut(     FacetCut[] calldata _diamondCut, </pre>

	<pre>         address _init,         bytes calldata _calldata     ) external override {         DiamondStorageLib.enforceIsContractOwner();         DiamondStorageLib.diamondCut(_diamondCut, _init, _calldata);     } </pre>
<b>Result/Recommendation</b>	We recommend upgrading the contracts in a single transaction or making the fallback function pausable for the duration of an upgrade.

#### 6.2.9 Missing Validation On amount

Severity: LOW

Status: **FIXED**

Code: NA

File(s) affected: Pool.sol

Update: Added in commit [#32062175462fa4acaf59138d270dbf8baa4eb960](#)

<b>Attack / Description</b>	In the deposit function the user can deposit numerous tokens in the Pool. However, there is no validation on the amount variable, a user can call this function with the value 0 and the user will lose gas fees without any impact.
<b>Code</b>	<p>Line 169 - 174 (Pool.sol)</p> <pre> function deposit(uint256 amount) public virtual nonReentrant {     _accumulateDepositInterest(msg.sender);      _transferToPool(msg.sender, amount);      _mint(msg.sender, amount);     _deposited[address(this)] += amount;     _updateRates(); </pre>

	<pre> emit Deposit(msg.sender, amount, block.timestamp); } </pre>
<b>Result/Recommendation</b>	Consider verifying that amount exceeds 0.

## 6.2.10 Possible DDoS In The Borrow

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: Pool.sol

Update: We agree that the probability of this event occurring is indeed very low.

On top of reasons mentioned in the report it's also worth noting that 256 bits can hold a maximum value of approximately  $1.1579 \times 10^{77}$  and at the same time currently supported tokens have 18 decimals only and the total supply of e.g. AVAX is 720 million tokens so there is not even enough supply of AVAX to come close to this scenario.

Should we need to integrate a token with some enormous number of decimals / totalSupply - we do have upgrade mechanisms in place to achieve that. For now we will just acknowledge this finding.

<b>Attack / Description</b>	<p>The borrow function, is incrementing the borrowed[msg.sender] and borrowed[address(this)] with the amount of value, if someone managed to borrow an amount with the value of <math>2^{256} - 1</math> the next borrower can't borrow, since the increment operation will overflow and the transaction will fail.</p> <p>The probability is very low, due to two reasons:</p> <ul style="list-style-type: none"> <li>- The user should have the ability to borrow all, without being liquidated.</li> <li>- The pool should have already the value, unless the attacker has provided the liquidity.</li> </ul>
<b>Code</b>	<p>Line 213 - 226 (Pool.sol)</p> <pre> function borrow(uint256 _amount) public virtual canBorrow nonReentrant {     require(IERC20(tokenAddress).balanceOf(address(this)) &gt;= _amount, "There is not enough funds in the pool to fund the loan"); } </pre>

	<pre>         _accumulateBorrowingInterest(msg.sender);          borrowed[msg.sender] += _amount;         borrowed[address(this)] += _amount;          _transferFromPool(msg.sender, _amount);          _updateRates();          emit Borrowing(msg.sender, _amount, block.timestamp);     } </pre>
<b>Result/Recommendation</b>	Consider verifying that the user has some MAX_BOUND for borrowing, to prevent the overflow.

#### 6.2.11 Missing Verification In Constructor For Duplicate Values

Severity: LOW

Status: **FIXED**

Code: NA

File(s) affected: RedstoneConfigManager.sol

Update: Added in commit [#24f0bfa4b31cc5b5c49c4d0b5c67180cf852b302](#)

<b>Attack / Description</b>	The constructor loops over the _trustedSigners array and adds each element to trusted- Signers using the function _addTrustedSigner. However, this function does not verify if the array contains duplicated elements.
<b>Code</b>	<p>Line 11 - 15 (RedstoneConfigManager.sol)</p> <pre> constructor(address[] memory _trustedSigners) {     for (uint256 i = 0; i &lt; _trustedSigners.length; i++) {         _addTrustedSigner(_trustedSigners[i]);     } } </pre>

	<pre>     }  Line 32 - 35 (RedstoneConfigManager.sol)  function _addTrustedSigner(address newSigner) private {     signerAuthorized[newSigner] = true;     trustedSigners.push(newSigner); } </pre>
<b>Result/Recommendation</b>	Consider adding a signerExists condition in the constructor to verify if a signer already exists before calling the _addTrustedSigner function.

## INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **5 Informational issues** in the code of the smart contract.

### 6.2.12 Floating Pragma Version Identified

Severity: INFORMATIONAL

Status: **FIXED**

Code: SWC-103

File(s) affected: ALL

Update: Added in [#009c16fab185e9c7a7c1dc02f87f61b701391f6f](#)

<b>Attack / Description</b>	It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
<b>Code</b>	e.g. Line 2 <pre>pragma solidity ^0.8.17;</pre>

<b>Result/Recommendation</b>	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.  i.e. Pragma solidity 0.8.17
------------------------------	--

### 6.2.13 Unnecessary Initializations

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: VariableUtilisationRatesCalculator.sol, UniswapV2DEXFacet.sol

Update: That is correct however in some cases being more explicit may serve for better readability.

<b>Attack / Description</b>	When a variable is declared in solidity, it gets initialized with type's default value.
<b>Code</b>	Line 17 (VariableUtilisationRatesCalculator.sol) <pre>uint256 public constant SLOPE_1 = 0;</pre> Line 35 (UniswapV2DEXFacet.sol) <pre>uint8 i = 0;</pre>
<b>Result/Recommendation</b>	There is no need to initialize a variable with the default value.

### 6.2.14 Function State Mutability Can Be Restricted To Pure

Severity: INFORMATIONAL

Status: **FIXED**

Code: NA



File(s) affected: SolvencyFacet.sol, VariableUtilisationRatesCalcuator.sol, SmartLoanLiquidationFacet.sol, PangolinDEXFacet.sol, TraderJoeDEXFacet.sol, UbeswapDEXFacet.sol, UbeswapIntermediary.sol,  
Update: Added in commit [#e7aeaddb868b9843648eb5a60cc7b9081b0abb5a](#)

Attack / Description	It is recommended to change the function state mutability to pure when the function does not read or modify the variables of the state.
Code	<p>Line 164 (SolvencyFacet.sol)  <code>function getMaxLtv() public view returns (uint256) {</code></p> <p>Line 81 (VariableUtilisationRatesCalcuator.sol)  <code>function calculateBorrowingRate(uint256 totalLoans, uint256 totalDeposits) external view override returns (uint256) {</code></p> <p>Line 50 (SmartLoanLiquidationFacet.sol)  <code>function getMaxLtv() public view returns (uint256) {</code></p> <p>Line 57 (SmartLoanLiquidationFacet.sol)  <code>function getMinLtvAfterLiquidation() public view returns (uint256) {</code></p> <p>Line 64 (SmartLoanLiquidationFacet.sol)  <code>function getMaxLiquidationBonus() public view returns (uint256) {</code></p> <p>Line 35 (PangolinDEXFacet.sol)  <code>function getExchangeIntermediaryContract() public override returns (address) {</code></p> <p>Line 35 (TraderJoeDEXFacet.sol)  <code>function getExchangeIntermediaryContract() public override returns (address) {</code></p> <p>Line 24 (UbeswapDEXFacet.sol)  <code>function getExchangeIntermediaryContract() public override returns (address) {</code></p>

	Line 19 (UbeswapIntermediary.sol) <pre>function getPath(address _token1, address _token2) internal override view returns (address[] memory) {</pre>
<b>Result/Recommendation</b>	It is recommended to change the function state mutability to pure when the function does not read or modify the variables of the state.

#### 6.2.15 Remove Existing Declaration

Severity: INFORMATIONAL

Status: **FIXED**

Code: NA

File(s) affected: UniswapV2Intermediary.sol

Update: Added in commit [#e7aeaddb868b9843648eb5a60cc7b9081b0abb5a](#)

<b>Attack / Description</b>	The amounts array is already declared in the returns statement.
<b>Code</b>	Line 53 (UniswapV2Intermediary.sol) <pre>uint256[] memory amounts = router.swapExactTokensForTokens(_exactSold, _minimumBought, getPath(_soldToken, _boughtToken), msg.sender, block.timestamp);</pre>
<b>Result/Recommendation</b>	It is recommended to remove the new declaration and perform only the amounts initialization.

#### 6.2.16 Remove Unused Local Variables

Severity: INFORMATIONAL

Status: **FIXED**

Code: NA

File(s) affected: SolvencyFacet.sol, SmartLoanLiquidationFacet.sol, UniswapV2DEXFacet.sol

Update: Added in commit [#e7aeaddb868b9843648eb5a60cc7b9081b0abb5a](#)

<b>Attack / Description</b>	There are some local variables that are declared
<b>Code</b>	<p>Line 108 (SolvencyFacet.sol)  <a href="#">uint256</a> valueOfStaked;</p> <p>Line 179 (SmartLoanLiquidationFacet.sol)  <a href="#">uint256</a> total = _getTotalValue();</p> <p>Line 99 (UniswapV2DEXFacet.sol)  TokenManager tokenManager = DeploymentConstants.getTokenManager();</p> <p>Line 141 (UniswapV2DEXFacet.sol)  TokenManager tokenManager = DeploymentConstants.getTokenManager();</p>
<b>Result/Recommendation</b>	it is recommended to either make use of them or simply remove them from the code.

## 6.3 SWC Attacks

ID	Title	Relationships	Test Result
<a href="#">SWC-131</a>	Presence of unused variables	<a href="#">CWE-1164: Irrelevant Code</a>	✓
<a href="#">SWC-130</a>	Right-To-Left-Override control character (U+202E)	<a href="#">CWE-451: User Interface (UI) Misrepresentation of Critical Information</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-129</a>	Typographical Error	<a href="#">CWE-480: Use of Incorrect Operator</a>	✓
<a href="#">SWC-128</a>	DoS With Block Gas Limit	<a href="#">CWE-400: Uncontrolled Resource Consumption</a>	✓
<a href="#">SWC-127</a>	Arbitrary Jump with Function Type Variable	<a href="#">CWE-695: Use of Low-Level Functionality</a>	✓
<a href="#">SWC-125</a>	Incorrect Inheritance Order	<a href="#">CWE-696: Incorrect Behavior Order</a>	✓
<a href="#">SWC-124</a>	Write to Arbitrary Storage Location	<a href="#">CWE-123: Write-what-where Condition</a>	✓
<a href="#">SWC-123</a>	Requirement Violation	<a href="#">CWE-573: Improper Following of Specification by Caller</a>	✓
<a href="#">SWC-122</a>	Lack of Proper Signature Verification	<a href="#">CWE-345: Insufficient Verification of Data Authenticity</a>	✓
<a href="#">SWC-121</a>	Missing Protection against Signature Replay Attacks	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-120</a>	Weak Sources of Randomness from Chain Attributes	<a href="#">CWE-330: Use of Insufficiently Random Values</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-119</a>	Shadowing State Variables	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-118</a>	Incorrect Constructor Name	<a href="#">CWE-665: Improper Initialization</a>	✓
<a href="#">SWC-117</a>	Signature Malleability	<a href="#">CWE-347: Improper Verification of Cryptographic Signature</a>	✓
<a href="#">SWC-116</a>	Timestamp Dependence	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-115</a>	Authorization through tx.origin	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-114</a>	Transaction Order Dependence	<a href="#">CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')</a>	✓
<a href="#">SWC-113</a>	DoS with Failed Call	<a href="#">CWE-703: Improper Check or Handling of Exceptional Conditions</a>	✓
<a href="#">SWC-112</a>	Delegatecall to Untrusted Callee	<a href="#">CWE-829: Inclusion of Functionality from Untrusted Control Sphere</a>	✓
<a href="#">SWC-111</a>	Use of Deprecated Solidity Functions	<a href="#">CWE-477: Use of Obsolete Function</a>	✓
<a href="#">SWC-110</a>	Assert Violation	<a href="#">CWE-670: Always-Incorrect Control Flow Implementation</a>	✓

ID	Title	Relationships	Test Result
<a href="#">SWC-109</a>	Uninitialized Storage Pointer	<a href="#">CWE-824: Access of Uninitialized Pointer</a>	✓
<a href="#">SWC-108</a>	State Variable Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓
<a href="#">SWC-107</a>	Reentrancy	<a href="#">CWE-841: Improper Enforcement of Behavioral Workflow</a>	✓
<a href="#">SWC-106</a>	Unprotected SELFDESTRUCT Instruction	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-105</a>	Unprotected Ether Withdrawal	<a href="#">CWE-284: Improper Access Control</a>	✓
<a href="#">SWC-104</a>	Unchecked Call Return Value	<a href="#">CWE-252: Unchecked Return Value</a>	✓
<a href="#">SWC-103</a>	Floating Pragma	<a href="#">CWE-664: Improper Control of a Resource Through its Lifetime</a>	✗
<a href="#">SWC-102</a>	Outdated Compiler Version	<a href="#">CWE-937: Using Components with Known Vulnerabilities</a>	✓
<a href="#">SWC-101</a>	Integer Overflow and Underflow	<a href="#">CWE-682: Incorrect Calculation</a>	✓
<a href="#">SWC-100</a>	Function Default Visibility	<a href="#">CWE-710: Improper Adherence to Coding Standards</a>	✓

## 6.4 Verify Claims

### Protocol related threats:

- Exploit through ERC20 methods
- Solvency - every method in a SmartLoan that could decrease its solvency has to check if it does not breach the solvency limit
- Oracle (PriceAware contracts) - important to check if data feeds could be manipulated by a user or third-party (contracts check signers of data packages)
- Upgradability - there are many places where proxy contracts are used. It's important to check if only admin wallets are able to upgrade the contracts and if they are correctly set.
- Wrong accounting of deposited/borrowed funds, accumulating deposits etc.
- Unauthorized party updating the list of whitelisted assets
- Unauthorized party accessing method when solvent
- Wrong handling of owner
- Wrong calculation of rates, especially near inflection points or in extreme pool utilizations like 0 or  $\geq 100\%$
- Unauthorized party changing configuration
- Wrong accounting of owners/loans lists
- Exploiting the admin ownership transfer proposal / execution
- Taking over the admin role / executing contractOwner-only methods (found in DiamondCutFacet.sol contract)
- Unauthorized funcSig=>facetAddress mapping edition
- ***In general the biggest possible threat*** would be if a bad actor would be able to (in any way) manipulate this contract to return an address of a malicious contract in a response to a **SmartLoanDiamondProxy (SmartLoan)** contract asking for an address to .delegatecall() to as this would mean execution of an arbitrary code in the context of user SmartLoan that holds custody over both his collateral as well as the borrowed funds.
- Wrong compounding of values
- Unauthorized party changing ownership
- Liquidation of a solvent account
- Any manipulation by a liquidator to exploit liquidated account
- Wrong evaluation of assets or debts
- Wrong staking/unstaking

- Wrong handling of reward assets
- Wrong handling of slippage
- Unauthorized addition/removal/update of methods' selectors=>addresses
- Unauthorized addition/removal of loan-owned assets

**Status:** tested and verified 

## 7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, one critical, one high, three medium, four low and seven informational issues have been found, after the manual and automated security testing.

We advise the DeltaPrime team to implement the recommendations contained in all 16 of our findings, to further enhance the overall security and readability.

Update (20/10/2022): The DeltaPrime team implemented necessary recommendations and responded to the issues very professional, with a deep understanding of solidity.



## 8. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive Web3 solutions. Their services include web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secured the smart contracts of 1Inch, POA Network, Unicrypt, LUKSO among numerous other top DeFi projects.

Chainsulting currently secures [\\$100 billion](#) in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: <https://chainsulting.de>

### How We Work



**1** -----

#### PREPARATION

Supply our team with audit ready code and additional materials



**2** -----

#### COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



**3** -----

#### AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



**4** -----

#### FIXES

Your development team applies fixes while consulting with our auditors on their safety.



**5** -----

#### REPORT

We check the applied fixes and deliver a full report on all steps done.