



# DeltaPrime

## Security Audit Report

June 19, 2024

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	About DeltaPrime . . . . .	3
1.2	Source Code . . . . .	3
<b>2</b>	<b>Overall Assessment</b>	<b>5</b>
<b>3</b>	<b>Vulnerability Summary</b>	<b>6</b>
3.1	Overview . . . . .	6
3.2	Security Level Reference . . . . .	7
3.3	Vulnerability Details . . . . .	8
<b>4</b>	<b>Conclusion</b>	<b>15</b>
<b>5</b>	<b>Appendix</b>	<b>16</b>
5.1	About AstraSec . . . . .	16
5.2	Disclaimer . . . . .	16
5.3	Contact . . . . .	17

---

# 1 | Introduction

## 1.1 About DeltaPrime

DeltaPrime protocol is a lending platform that allows under-collateralized borrowing. The key innovation is to enable fund lending to a special-purpose smart contract rather than a personal account. This contract ensures solvency by automatically enforcing a series of checks for every activity. Additionally, a decentralized liquidation system mitigates insolvency risk, allowing anyone to forcibly repay part of the loan if external factors cause fluctuations in asset prices.

## 1.2 Source Code

The audit scope covers the code changes in below PR:

- <https://github.com/DeltaPrimeLabs/deltaprime-primeloans/pull/290>

And this is the final repository and commit hash representing all fixes implemented for the issues identified in the audit:

- <https://github.com/DeltaPrimeLabs/deltaprime-primeloans/tree/audit-fixes>
- CommitID: 1d4937c

---

The following are the SHA256 hashes of all the audited files:

Table 1.1: SHA256 of Audited Files

Contract	SHA256
DepositSwapAvalanche.sol	8d6a7d59cce38c69c7a19e73c39a84fd8dc8a8dc8f16fa3eb2d3896d47c2d454
DepositSwapArbitrum.sol	e6172af14e3d6fff41d3aa093750dd394616aa1778fa1aa8a7eb79e8409b73fd
OnlyOwnerOrInsolvent.sol	9273f7c13986965b2b0d11910f0c08ef7c209b8128a16f7f7848a3e71f76cfed
TokenManager.sol	73e1a5e4e3e69ed6505181c8194b8b9e26ec448f838d4bfa0e129ad37778164f
facets/AssetsOperationsFacet.sol	81699f90a0859d6afb4b35c6e78d8c687f13c3fee0f9307ece98490e4344bc84
facets/GmxV2CallbacksFacet.sol	4d9f4192134d5b26e921613dec67dba491cbc9e21ac3420201bc7c061d4cf191
facets/GmxV2Facet.sol	28ee0969e5d3044a03dea05b983f4b0f20167399e9f3de173267e8cc7d8feba3
facets/ParaSwapFacet.sol	caa3afc5226be2c73a7ea167e94d956ff281ddca498bb1dbff3a2a1854789bc6
facets/SmartLoanLiquidationFacet.sol	a9d2d3d36fa5d8b7229581a21d1fb781fed6bdba478ad3c2844ba1e4450e8037
facets/TraderJoeV2Facet.sol	241337391784be4f11c461ec4411695bebf308d931e49f15a9003516c8f1151a
facets/arbitrum/BeefyFinanceArbitrumFacet.sol	f53bac24ee50aab2e3190fe4b4cc03693c8fbd73d585c79adc679b02498583c4
facets/avalanche/GLPFacet.sol	7f7f650a106473eea4105337609c8d95e48bae291a8440842932178f3ef9f258
facets/arbitrum/PenpieFacet.sol	4869721de03c70f89a6b5cdd827191efc9b99c9db4fbc6de1453cb994345f163
facets/avalanche/BalancerV2Facet.sol	02344ddfc4b3b3b611a003fe73594756160009c1f4f2c730af9cb58bc007d59
facets/avalanche/CaiFacet.sol	33f1bc7bc3687985a8967b121962c7b441ae8f13c5fa3b118720df2c0c117c66
facets/avalanche/GogoPoolFacet.sol	72fe6820cc592138c41f38435b77fde4eaf9348ea5a622ebdb9ef4d26381096
facets/avalanche/UniswapV3Facet.sol	5d664e25cdc9f98fd0d2a4e96ed13e55f861f87ed762fc8d20d7f3ec9a6c6439
facets/avalanche/YieldYakFacet.sol	89d86962e27727a39a3e62c4c5450dc093591d0eae00125c0008358b641d2aba
facets/avalanche/YieldYakSwapFacet.sol	ed7ae672f8b119e408196e577f495386ce35900e96c2758fb186e237ef6bfde1
contracts/lib/DiamondStorageLib.sol	344017a0f42e055b8dbb8aa288adbb40ea66dcc8dbff505e921af710d3219d66

---

## 2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `DeltaPrime` protocol. Throughout this audit, we identified a total of 6 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	3	2	-	1
Low	3	-	-	3
Informational	-	-	-	-
Undetermined	-	-	-	-

---

## 3 | Vulnerability Summary

### 3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

- M-1** [Incorrect Array Size in GmxV2Facet::\\_withdraw\(\)](#)
- M-2** [Bypass of Exposure Check by Direct Transferring-in Aseets](#)
- M-3** [Potential Risks Associated with Centralization](#)
- L-1** [Inconsistent Price Precision in decreaseLiquidityUniswapV3\(\)](#)
- L-2** [Potential Denial-of-Service in liquidate\(\)](#)
- L-3** [Revised Event Emitted in removeOwnedAsset\(\)](#)

---

## 3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

---

## 3.3 Vulnerability Details

### [M-1] Incorrect Array Size in GmxV2Facet::\_withdraw()

Target	Category	IMPACT	LIKELIHOOD	STATUS
GmxV2Facet.sol	Coding Practices	Medium	Medium	<a href="#">Addressed</a>

The `GmxV2Facet` contract is designed to facilitate investment in `GMX`, allowing users to deposit or withdraw assets. During the withdrawal operation, a solvency check is applied, which requires reading and storing the prices of three tokens: `gmToken`, `longToken` and `shortToken`.

The prices of these tokens are stored in an array `tokenPrices`, which is incorrectly defined to have a size of 2 (line 162). This size constraint causes an overflow when attempting to store the prices of all three tokens, leading to potential errors in the contract execution.

#### GmxV2Facet::\_withdraw()

```
161 if(msg.sender == DiamondStorageLib.contractOwner()){
162     uint256[] memory tokenPrices = new uint256[] (2);
163     {
164         bytes32[] memory tokenSymbols = new bytes32[] (3);
165         tokenSymbols[0] = tokenManager.tokenAddressToSymbol(longToken);
166         tokenSymbols[1] = tokenManager.tokenAddressToSymbol(shortToken);
167         tokenSymbols[2] = tokenManager.tokenAddressToSymbol(gmToken);
168         tokenPrices = getPrices(tokenSymbols);
169     }
170     require(isWithinBounds(
171         tokenPrices[2] * gmAmount / 10**IERC20Metadata(gmToken).decimals(),
172         // Deposit Amount In USD
173         tokenPrices[0] * minLongTokenAmount / 10**IERC20Metadata(longToken).
174             decimals()
175         + tokenPrices[1] * minShortTokenAmount / 10**IERC20Metadata(shortToken).
176             decimals()) // Output Amount In USD
177         , "Invalid min output value");
178     ...
179 }
```

**Remediation** The `tokenPrices` array should be defined with a size of 3 to accommodate the prices of `gmToken`, `longToken` and `shortToken`.



## [M-2] Bypass of Exposure Check by Direct Transferring-in Aseets

Target	Category	IMPACT	LIKELIHOOD	STATUS
AssetsOperationsFacet.sol	Business Logic	Medium	Medium	Acknowledged

The DeltaPrime protocol incorporates an exposure check for each asset group to limit the total amount of assets within each group to a specified capacity. The exposure of each group is updated whenever an asset is funded to or withdrawn from the loan. This is intended to manage risk and ensure proper asset management within the protocol.

To elaborate, we show below the code snippet of the `AssetsOperationsFacet::fund()` function, which is used to fund assets into the loan. It pulls assets from the caller (line 39) and invokes the `_increaseExposure()` function to update and check the exposure (line 42).

### AssetsOperationsFacet::fund()

```
35 function fund(bytes32 _fundedAsset, uint256 _amount) public virtual nonReentrant{
36     IERC20Metadata token = getERC20TokenInstance(_fundedAsset, false);
37     _amount = Math.min(_amount, token.balanceOf(msg.sender));

39     address(token).safeTransferFrom(msg.sender, address(this), _amount);

41     ITokenManager tokenManager = DeploymentConstants.getTokenManager();
42     _increaseExposure(tokenManager, address(token), _amount);

44     emit Funded(msg.sender, _fundedAsset, _amount, block.timestamp);
45 }
```

However, when reviewing the `SolvencyFacetProd::_getTWVOwnedAssets()` function which is used to calculate the total value in the loan, we notice that it uses the `token.balanceOf(address(this))` (line 297) as the total balance of the token. This allows the loan owner to bypass the exposure check by directly transferring assets into the loan, instead of using the `AssetsOperationsFacet::fund()` function. Below is the relevant code snippet:

### SolvencyFacetProd::\_getTWVOwnedAssets()

```
287 function _getTWVOwnedAssets(AssetPrice[] memory ownedAssetsPrices) internal view
    returns (uint256) {
288     bytes32 nativeTokenSymbol = DeploymentConstants.getNativeTokenSymbol();
289     ITokenManager tokenManager = DeploymentConstants.getTokenManager();

291     uint256 weightedValueOfTokens = ownedAssetsPrices[0].price * (address(this).
        balance - msg.value) * tokenManager.debtCoverage(tokenManager.
        getAssetAddress(nativeTokenSymbol, true)) / (10 ** 26);
```

```

293     if (ownedAssetsPrices.length > 0) {
294
295         for (uint256 i = 0; i < ownedAssetsPrices.length; i++) {
296             IERC20Metadata token = IERC20Metadata(tokenManager.getAssetAddress(
297                 ownedAssetsPrices[i].asset, true));
298             weightedValueOfTokens = weightedValueOfTokens + (ownedAssetsPrices[i].
299                 price * token.balanceOf(address(this)) * tokenManager.debtCoverage(
300                     address(token)) / (10 ** token.decimals() * 1e8));
301         }
302     }
303     return weightedValueOfTokens;
304 }

```

**Remediation** It is recommended to revisit the protocol design to improve the exposure check mechanism.

**Response By Team** This issue has been acknowledged by the team.

### [M-3] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	High	Low	Mitigated

In the DeltaPrime protocol, the existence of the privileged owner account introduces centralization risks, as it holds significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged account.

#### Example Privileged Operations

```

61 function whitelistLiquidators(address[] memory _liquidators) external onlyOwner
62 {
63     DiamondStorageLib.LiquidationStorage storage ls = DiamondStorageLib.
64         liquidationStorage();
65
66     for(uint i; i<_liquidators.length; i++){
67         ls.canLiquidate[_liquidators[i]] = true;
68         emit LiquidatorWhitelisted(_liquidators[i], msg.sender, block.timestamp);
69     }
70 }
71
72 function delistLiquidators(address[] memory _liquidators) external onlyOwner {
73     DiamondStorageLib.LiquidationStorage storage ls = DiamondStorageLib.
74         liquidationStorage();
75     for(uint i; i<_liquidators.length; i++){

```

```

74         ls.canLiquidate[_liquidators[i]] = false;
75         emit LiquidatorDelisted(_liquidators[i], msg.sender, block.timestamp);
76     }
77 }

```

**Remediation** To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged account. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

**Response By Team** This issue has been mitigated by implementing a multisig and timelock mechanism to manage the owner.

## [L-1] Inconsistent Price Precision in decreaseLiquidityUniswapV3

Target	Category	IMPACT	LIKELIHOOD	STATUS
UniswapV3Facet.sol	Coding Practices	Medium	Low	<a href="#">Addressed</a>

The UniswapV3Facet contract provides the `decreaseLiquidityUniswapV3()` function for users to decrease liquidity from Uniswap V3. To protect users' funds from potential loss due to price manipulation, it applies a slippage check before the liquidity decrease. The slippage check calculates the token price in the pool and compares it with the oracle price to ensure the difference is within the allowed threshold.

To elaborate, we show below the code snippet of the `decreaseLiquidityUniswapV3()` function. It first calculates the oracle price (`oraclePrice`), which has a hardcoded 18 decimals (line 204). Then it calculates the token price in the pool (`poolPrice`) which has the same decimals as `token0` (line 205), i.e., `IERC20Metadata(token0Address).decimals()`.

Specifically, if `token0` doesn't have 18 decimals, the `poolPrice` doesn't have 18 decimals either. Consequently, the comparison between the `poolPrice` and the `oraclePrice` will not have a correct result. Based on this, it's recommended to use the same decimals for both the `poolPrice` and the `oraclePrice`.

### UniswapV3Facet::decreaseLiquidityUniswapV3()

```

193 (uint160 poolSqrtPrice,,,,,) = IUniswapV3Pool(poolAddress).slot0();
194 uint256 sqrtPoolPrice = UniswapV3IntegrationHelper.sqrtPriceX96ToSqrtUint(
    poolSqrtPrice, IERC20Metadata(token0Address).decimals());

196 bytes32[] memory symbols = new bytes32[](2);

198 symbols[0] = token0;

```

```

199 symbols[1] = token1;

201 uint256[] memory prices = getOracleNumericValuesFromTxMsg(symbols);

203 uint256 oraclePrice = prices[0] * 1e18 / prices[1];
204 uint256 poolPrice = sqrtPoolPrice * sqrtPoolPrice / 10 ** IERC20Metadata(
    token1Address).decimals();

206 if (oraclePrice > poolPrice) {
207     if ((oraclePrice - poolPrice) * 1e18 / oraclePrice >
        ACCEPTED_UNISWAP_SLIPPAGE) revert SlippageTooHigh();
208 } else {
209     if ((poolPrice - oraclePrice) * 1e18 / oraclePrice >
        ACCEPTED_UNISWAP_SLIPPAGE) revert SlippageTooHigh();
210 }

```

**Remediation** Revisit the slippage check in the `decreaseLiquidityUniswapV3()` function and use the same decimals for both the `poolPrice` and the `oraclePrice`.

## [L-2] Potential Denial-of-Service in `liquidate()`

Target	Category	IMPACT	LIKELIHOOD	STATUS
SmartLoanLiquidationFacet.sol	Business Logic	Low	Low	Addressed

The `liquidate()` function is designed to allow whitelisted liquidators to address insolvent loans. This function enables a liquidator to specify the amount of debt they intend to repay. If the loan's token balance is insufficient to cover this amount, the function pulls the required excess (referred to as `supplyAmount`) from the liquidator. The final repay amount (`repayAmount`) is the smaller one of the debt amount and the to-repay amount (line 172). After the repayment, it updates the exposure of the token with the token balance change in the loan, i.e., `repayAmount - supplyAmount` (line 181).

However, we noticed that the `repayAmount` and the `supplyAmount` are derived from user inputs, this dependency introduces a potential risk where `repayAmount` may not be greater than `supplyAmount`. If this occurs, the expression `(repayAmount - supplyAmount)` could underflow, causing the liquidation process to revert. Our analysis shows that the exposure should be decreased if `(repayAmount >= supplyAmount)` and increased otherwise.

### liquidate

```

154 for (uint256 i = 0; i < config.assetsToRepay.length; i++) {
155     IERC20Metadata token = IERC20Metadata(tokenManager.getAssetAddress(config.
        assetsToRepay[i], true));

```

```

157     uint256 balance = token.balanceOf(address(this));
158     uint256 supplyAmount;

160     if (balance < config.amountsToRepay[i]) {
161         supplyAmount = config.amountsToRepay[i] - balance;
162     }

164     if (supplyAmount > 0) {
165         address(token).safeTransferFrom(msg.sender, address(this), supplyAmount);
166         // supplyAmount is denominated in token.decimals(). Price is denominated in
167         // 1e8. To achieve 1e18 decimals we need to multiply by 1e10.
168         suppliedInUSD += supplyAmount * cachedPrices.assetsToRepayPrices[i].price *
169             10 ** 10 / 10 ** token.decimals();
170     }

172     Pool pool = Pool(tokenManager.getPoolAddress(config.assetsToRepay[i]));

174     uint256 repayAmount = Math.min(pool.getBorrowed(address(this)), config.
175         amountsToRepay[i]);

176     address(token).safeApprove(address(pool), 0);
177     address(token).safeApprove(address(pool), repayAmount);

179     // repayAmount is denominated in token.decimals(). Price is denominated in 1e8.
180     // To achieve 1e18 decimals we need to multiply by 1e10.
181     repaidInUSD += repayAmount * cachedPrices.assetsToRepayPrices[i].price * 10 **
182         10 / 10 ** token.decimals();

184     pool.repay(repayAmount);
185     _decreaseExposure(tokenManager, address(token), repayAmount - supplyAmount);
186     ...
187 }

```

**Remediation** Properly revise the exposure update process and accommodate all possible values of the repayAmount and supplyAmount.

### [L-3] Revised Event Emitted in removeOwnedAsset()

Target	Category	IMPACT	LIKELIHOOD	STATUS
DiamondStorageLib.sol	Coding Practices	Low	Low	Addressed

In Ethereum smart contracts, it is crucial to trigger events using the `emit` keyword because events allow smart contracts to communicate with the external world and provide a mechanism for DApp frontends to listen for changes in contract state.

The `removeOwnedAsset()` function, as shown in the code below, is used to remove an owned asset of a user. However, it emits the wrong event, `OwnedAssetAdded()`, which is intended for adding an

---

owned asset. It should emit the `OwnedAssetRemoved()` event.

#### **removeOwnedAsset()**

```
278 function removeOwnedAsset(bytes32 _symbol) internal {  
279     SmartLoanStorage storage sls = smartLoanStorage();  
280     EnumerableMap.remove(sls.ownedAssets, _symbol);  
  
282     emit OwnedAssetAdded(_symbol, block.timestamp);  
283 }
```

**Remediation** Emit the correct `OwnedAssetRemoved()` event in the function.

---

## 4 | Conclusion

In this audit, we have reviewed the smart contracts of the `DeltaPrime` protocol, a lending platform that allows under-collateralized borrowing. The key innovation is to enable fund lending to a special-purpose smart contract rather than a personal account. This contract ensures solvency by automatically enforcing a series of checks for every activity. Additionally, a decentralized liquidation system mitigates insolvency risk, allowing anyone to forcibly repay part of the loan if external factors cause fluctuations in asset prices.

The current code base is well-structured and neatly organized. The identified issues have been promptly confirmed and fixed.

---

## 5 | Appendix

### 5.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

### 5.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.



---

## 5.3 Contact

<b>Name</b>	AstraSec Team
<b>Phone</b>	+86 176 2267 4194
<b>Email</b>	contact@astrasec.ai
<b>Twitter</b>	<a href="https://twitter.com/AstraSecAI">https://twitter.com/AstraSecAI</a>