

# SMART CONTRACT SECURITY

v 1.0

Date: 24-09-2025

Prepared For: Meta Pool



## About BlockApex

Founded in early 2021 BlockApex is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on [Twitter](#) and explore our [GitHub](#). For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our [Contact](#) page at our website , or reach out to us via email at [hello@blockapex.io](mailto:hello@blockapex.io).

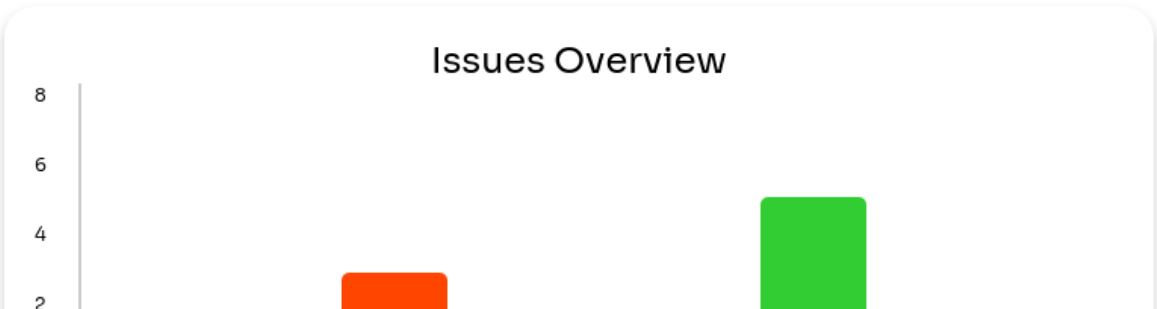
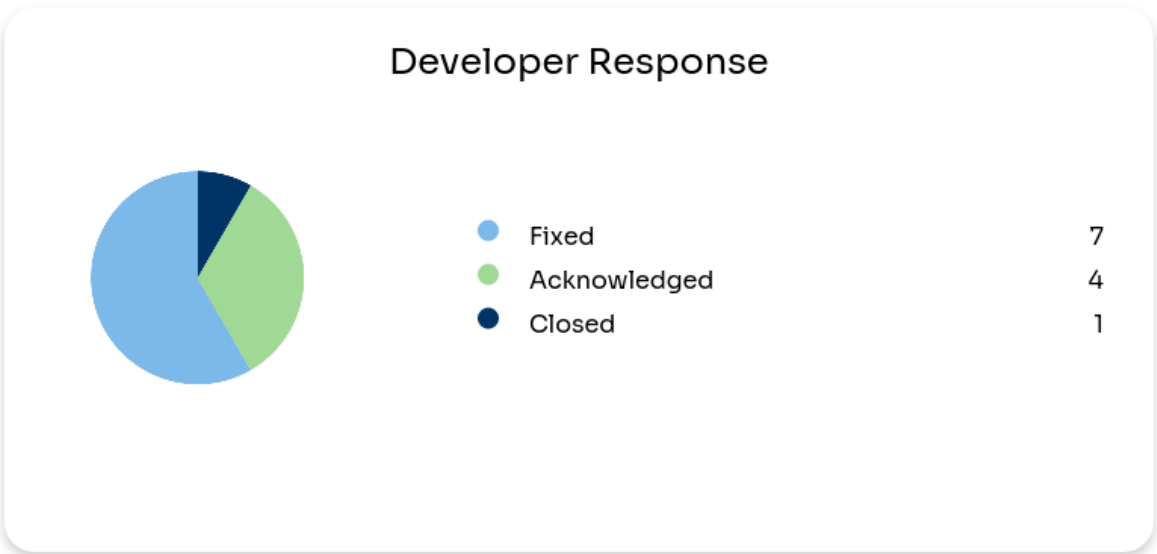
## Contents

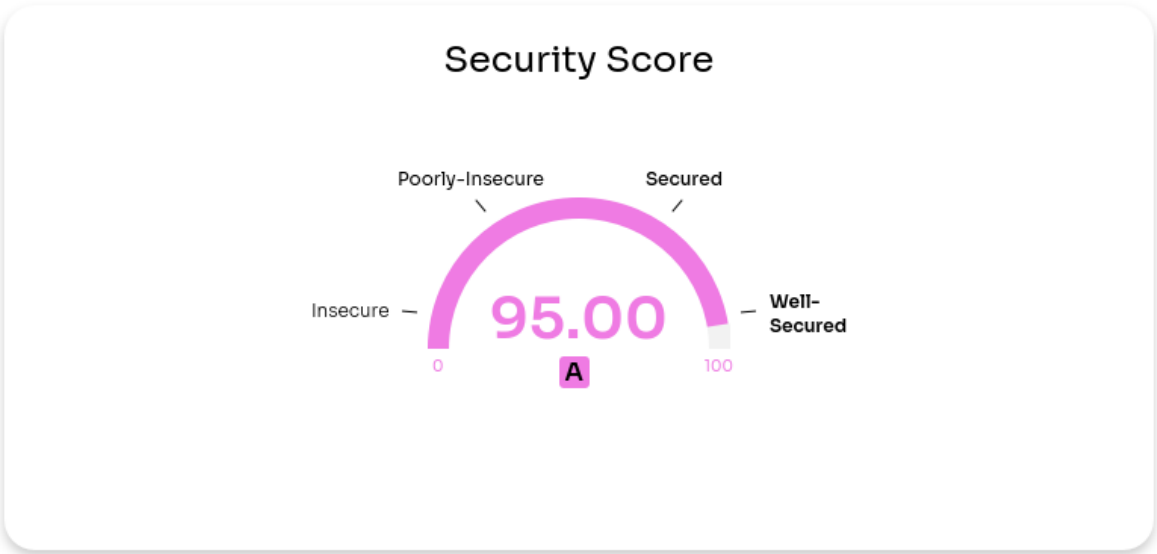
<b>1</b>	<b>Executive Summary</b>	<b>4</b>
1.1	Scope . . . . .	6
1.1.1	In Scope . . . . .	6
1.1.2	Out of Scope . . . . .	6
1.2	Methodology . . . . .	6
1.3	Status Descriptions . . . . .	8
1.4	Summary of Findings Identified . . . . .	9
<b>2</b>	<b>Findings and Risk Analysis</b>	<b>10</b>
2.1	Verde Token \$1 USD Peg Assumption Creates Systemic Protocol Risk . . . . .	10
2.2	No Liquidation Incentives for the Liquidators Above 100% LTV Creates Bad Debt Risk for the Protocol . . . . .	12
2.3	Missing _decimalsOffset() Override Leads to Potential Inflation Attack in ERC4626 Vault	14
2.4	Missing Validation Oracle on latestRoundData() Exposes Protocol to Stale or Invalid Prices	15
2.5	Direct ERC4626 Implementation Exposes Users to Multiple Vault Risks . . . . .	17
2.6	Incomplete Pause Functionality will get borrowers liquidated unfairly . . . . .	18
2.7	Missing debtCap Check in Interest Accrual . . . . .	19
2.8	Unsafe LTV Reduction Without Active Loan Check . . . . .	20
2.9	No Emergency Liquidation Pause During Collateral Asset Depegging Events . . . . .	21
2.10	Liquidators can Delay Liquidations to Maximize Profits in Higher Tiers . . . . .	22
2.11	Self-Liquidation Using Flash Loans to Minimize Losses . . . . .	23
2.12	Interest Accrual Results in Compounding (APY vs. APR) . . . . .	24

# 1 Executive Summary

A team of 2 Auditors performed the Filtered Audit,where the codebase was audited individually by both auditors. After a thorough and rigorous manual testing process involving line by line code review for bugs related to Rust and Near Smart Contracts. All the raised flags were manually reviewed and tested to identify any false positives.

The Audit resulted in 6 Highs, 2 medium, 5 low and 2 Info Severity Issues. The codebase received an overall security score of 95 out of 100 points.





## 1.1 Scope

### 1.1.1 In Scope

**Repository** <https://github.com/Meta-Pool/stable-verde/releases/tag/v0.4.0>

**Commit Hash** 62b94c2404f8d7302d1daf2b42f1be07bd352ce3

### 1.1.2 Out of Scope

Anything not mentioned in the Scope.

## 1.2 Methodology

The codebase was audited using a filtered audit technique. A band of two (2) auditors scanned the codebase in an iterative process for a time spanning 2.5 Weeks. Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations and security design patterns, code styles and best practices.

**Project Overview** MetaPool Stable-Verde a decentralized lending and borrowing protocol centered around the VERDE token, a USD-pegged stablecoin issued by Meta Pool. The core architecture allows users to deposit a yield-bearing collateral token (mtETH) and borrow VERDE against it, governed by loan-to-value (LTV) ratios and debt ceilings.

The protocol supports:

- **Collateralized Borrowing:** Users deposit a yield-bearing asset (e.g., mtETH) as collateral to borrow VERDE, with borrowing power governed by configurable loan-to-value (LTV) and liquidation thresholds.
- **Interest Accrual:** Debt accrues interest on a daily basis through the `_accrue()` function, which applies uniform updates to all borrowers once per day. The design intentionally avoids per-second accrual to reduce gas overhead. To address intraday timing effects, the protocol roadmap includes weekly or monthly amortized repayment plans with fixed installments, rather than continuous compounding.
- **Staking (stVERDE):** Users can stake their VERDE into a vault (stVERDE) built on the ERC-4626 standard to earn yield. Yield is sourced from treasury inflows such as protocol fees and borrow interest.
- **Token Swapping:** A swap module allows users to exchange between VERDE and a supported stablecoin (e.g., USDT or USDC), with price conversion based on oracle feeds. Swaps are subject to configurable fees and liquidity caps, and can be routed to designated receivers.
- **Treasury and Yield Distribution:** A treasury vault accumulates VERDE from interest and fees, which can be distributed to staking contracts or other destinations to support protocol incentives and sustainability.
- **Risk Controls and Configurability:** Administrators can adjust parameters like LTV, APR caps, debt ceilings, and fees. Security mechanisms include safe math, allowance management, and interest rate bounds.
- **Oracle Design:** The oracle applies a 5% maximum daily price change threshold to submitted updates, ensuring stability and protection against outlier feeds. An authorized keeper bot sources data from multiple providers and only pushes on-chain updates when within bounds. If volatility exceeds 5%, the update is flagged for manual review to safeguard against bad data.

The system leverages ERC-20 and ERC-4626 standards, includes precision handling for daily interest compounding, and integrates with a treasury component for managing yield distribution. The contracts aim to offer a permissionless, capital-efficient, and secure stablecoin lending ecosystem for the Meta Pool community.

### 1.3 Status Descriptions

**Acknowledged:** The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

**Fixed:** The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

**Closed:** This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.



## 1.4 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	HIGH	Verde Token \$1 USD Peg Assumption Creates Systemic Protocol Risk	FIXED
#2	HIGH	No Liquidation Incentives for the Liquidators Above 100% LTV Creates Bad Debt Risk for the Protocol	FIXED
#3	HIGH	Missing _decimalsOffset() Override Leads to Potential Inflation Attack in ERC4626 Vault	FIXED
#4	MEDIUM	Missing Validation Oracle on latestRoundData() Exposes Protocol to Stale or Invalid Prices	FIXED
#5	MEDIUM	Direct ERC4626 Implementation Exposes Users to Multiple Vault Risks	ACKNOWLEDGED
#6	LOW	Incomplete Pause Functionality will get borrowers liquidated unfairly	FIXED
#7	LOW	Missing debtCap Check in Interest Accrual	CLOSED
#8	LOW	Unsafe LTV Reduction Without Active Loan Check	ACKNOWLEDGED

S.NO	SEVERITY	FINDINGS	STATUS
#9	LOW	No Emergency Liquidation Pause During Collateral Asset Depegging Events	FIXED
#10	LOW	Liquidators can Delay Liquidations to Maximize Profits in Higher Tiers	FIXED
#11	INFO	Self-Liquidation Using Flash Loans to Minimize Losses	ACKNOWLEDGED
#12	INFO	Interest Accrual Results in Compounding (APY vs. APR)	ACKNOWLEDGED

## 2 Findings and Risk Analysis

### 2.1 Verde Token \$1 USD Peg Assumption Creates Systemic Protocol Risk

**Severity:** High

**Status:** Fixed

**Location:**

BorrowVerdeV1.sol SwapVerde.sol

**Description:** The protocol operates under the hardcoded assumption that Verde token always equals \$1 USD, despite stablecoins being susceptible to depegging events during market stress. This assumption is embedded throughout the codebase via conversion constants and exchange rate calculations:

```
1 // In BorrowVerdeV1.sol - assumes Verde = $1 for collateral conversions
2 _collat2VerdeConversionConstant = 10 ** (_collatToUsdOracle.decimals() + 18 - _verdeToken.
   decimals());
3
4 function fromCollat2VERDE(uint256 _amount) internal view returns (uint256) {
5     return _amount.mulDiv(_collatPriceUsd(), _collat2VerdeConversionConstant, Math.
       Rounding.Floor);
6 }
7
8 // In SwapVerde.sol - assumes Verde = $1 for stablecoin swaps
9 _stable2VerdeConversionConstant = 10 ** (_stableToUsdOracle.decimals() + 6 - _verdeToken.
   decimals());
10
11 function fromStable2VERDE(uint256 _amount) internal view returns (uint256) {
12     (, int256 price,,) = stableToUsdOracle.latestRoundData();
13     return _amount.mulDiv(_unsigned256(price), _stable2VerdeConversionConstant, Math.
       Rounding.Ceil);
14 }
```

All collateral valuations, LTV calculations, liquidation thresholds, and swap rates are calculated assuming Verde maintains perfect \$1 parity, creating systemic vulnerabilities when this assumption breaks.

**Recommendation:** Implement dynamic Verde pricing that reflects real market conditions rather than assuming a fixed \$1 peg.

#### Developer Response:

**1. 1:1 swap rate:** Originally we priced swaps at stable\_price (to guard against tiny marketprice deviations), but feedback was clear: users expect a true peg. Weve now switched to strict 1:1 mintandburn for all stable assetsjust like DAIs PSMso you always get exactly one protocol stable token per USDC (or USDT) in, with zero slippage and zero fee. ##### 2. Peg source: With 1:1 mint/burn, the peg is now

anchored directly to the onchain deposit of \$1 tokens, not to any external market quote. That matches DAIs model and removes any residual basispoint drift.

## 2.2 No Liquidation Incentives for the Liquidators Above 100% LTV Creates Bad Debt Risk for the Protocol

**Severity:** High

**Status:** Fixed

**Location:**

BorrowVerdeV1.sol

**Description:** The protocol's liquidation mechanism lacks proper incentives when LTV exceeds 100%, creating a critical bad debt scenario. The current tiered liquidation system in `_getLiquidationDeal()` provides the entire collateral to liquidators in exchange for paying the full debt when LTV 90%:

```
1 function _getLiquidationDeal(uint256 _debt, uint256 _collateral) internal
2 view returns (uint256 _requiredVerde, uint256 _collateralOut) {
3     uint256 realLtvBp = _getLoanToValueRatioBp(_debt, _collateral);
4     // Tier 3: LTV greater than 90.00%.
5     if (realLtvBp gret= 9000) {
6         _requiredVerde = _debt; // Liquidator pays FULL debt
7         _collateralOut = _collateral; // Liquidator gets ALL collateral
8     }
9     // ... other tiers
10 }
```

However, when LTV exceeds 100%, the debt value becomes greater than the collateral value, eliminating any economic incentive for liquidators to act.

**Recommendation:** Consider implementing automated liquidation mechanisms, treasury-backed incentives, or a Dutch auction system to ensure positions never remain unliquidated above 100% LTV, preventing bad debt accumulation and maintaining Verde token stability.

```
1 // Option 1: Treasury subsidy for underwater positions
2 if (realLtvBp gret= 10000) {
3     uint256 shortfall = _debt - fromCollat2VERDE(_collateral);
4     _requiredVerde = fromCollat2VERDE(_collateral); // Pay only collateral value
5     _collateralOut = _collateral;
6     // Treasury covers the shortfall to incentivize liquidation
7     _mintInterests(shortfall); // Mint to cover bad debt
8 }
9
10 // Option 2: Automated liquidation system
11 function _emergencyLiquidation(address _account) internal {
12     // Automatically liquidate when LTV gret 100% using protocol reserves
13     // Or implement a Dutch auction system for underwater collateral
14 }
```

### Developer Response:

**1. OffChain Automated Liquidation** MetaPool has deployed a dedicated bot that continuously monitors onchain loan positions and triggers liquidation transactions as soon as any borrowers LTV

exceeds the protocols liquidation threshold to minimize the underwater risk. ##### 2. Safety Fund Backstop The bot is provisioned from a treasury maintained safety fund (security last debt buyer) that covers any shortfall between debt and collateral value in underwater scenarios. In the event that collateral value < debt principal, the safety fund tops up the difference, ensuring no bad debt is left onchain.

## 2.3 Missing `_decimalsOffset()` Override Leads to Potential Inflation Attack in ERC4626 Vault

**Severity:** High

**Status:** Fixed

**Location:**

StakedVerdeV1.sol

**Description:** The StakedVerdeV1 contract is implemented as a wrapper around a 6-decimal stablecoin (Verde Token), using OpenZeppelins ERC4626 vault standard. However, the implementation omits an important override of the `_decimalsOffset()` function introduced in OpenZeppelin v4.9. This function determines the internal virtual asset/share scaling used by the vault to defend against donation/inflation attacks. When left as the default (0), the vault does not simulate any pre-existing assets or shares. This opens the door for attackers to manipulate the share price by donating tokens directly to the vault, skewing the total assets without minting proportional shares. This directly affects honest users who deposit afterward, as they may receive little or no shares for their legitimate deposits.

**Recommendation:** overriding `_decimalsOffset()` based on the decimals of the underlying asset:

```
1 function _decimalsOffset() internal view override returns (uint8) {
2     return 12; // For 6-decimal stablecoins
3 }
```

## 2.4 Missing Validation Oracle on latestRoundData() Exposes Protocol to Stale or Invalid Prices

**Severity:** Medium

**Status:** Fixed

**Location:**

contracts/SwapVerde.sol contracts/BorrowVerdeV1.sol

**Description:** The protocol uses Oracles latestRoundData() to fetch price feeds for critical operations like token swaps and collateral valuation, but fails to check whether the returned price is stale. Specifically, it does not verify the updatedAt or startedAt timestamps. This oversight allows outdated prices to be used in economic calculations, potentially leading to swaps at incorrect rates or over/under-collateralized positions. If the oracle stops updating (e.g., due to downtime or misconfiguration), users may unknowingly interact with stale prices, exposing the protocol to financial risk and exploitation.

In SwapVerde.sol:

```
1 function fromStable2Verde(...) {
2     (, int256 price,,, ) = stableToUsdOracle.latestRoundData(); // No checks
3     ...
4 }
5
6 function fromVerde2Stable(...) {
7     (, int256 price,,, ) = stableToUsdOracle.latestRoundData(); // No checks
8     ...
9 }
```

In BorrowVerdeV1.sol:

```
1 function _collatPriceUsd() private view returns (uint256) {
2     (, int256 price,,, ) = collatToUsdOracle.latestRoundData(); // No checks
3     return uint256(price);
4 }
```

These functions do not guard against stale or invalid data (e.g., price = 0, or timestamps are outdated), which can compromise protocol integrity.

**Recommendation:** \* To prevent the use of stale or invalid price data, implement a function that enforces explicit checks on the Chainlink feeds freshness and validity. \* Introduce a staleThreshold (e.g., 24 hour) as a configurable parameter.

```
1 function _getFreshPrice(AggregatorV3Interface oracle) internal view returns (uint256) {
2     (uint80 roundId, int256 price, , uint256 updatedAt, uint80 answeredInRound) = oracle.
        latestRoundData();
3
4     require(price gret 0, &quot;Invalid price&quot;);
5
6     require(updatedAt != 0 &amp;&amp; block.timestamp - updatedAt &lt;= staleThreshold, &
        quot;Stale price&quot;);
7 }
```

```
8   return uint256(price);  
9 }
```



## 2.5 Direct ERC4626 Implementation Exposes Users to Multiple Vault Risks

**Severity:** Medium

**Status:** Acknowledged

**Location:**

StakedVerdeV1.sol

**Description:** The `StakedVerdeV1` contract directly inherits from OpenZeppelin's `ERC4626` without implementing additional safety measures, exposing users to multiple vault-related risks. The contract serves as a liquid staking solution for Verde tokens but lacks critical protections that sophisticated vault implementations typically include:

```
1  contract StakedVerdeV1 is ERC4626 {
2      constructor(IERC20 _asset) ERC4626(_asset) ERC20("Staked Meta Pool USD Stablecoin", "stVERDE") {}
3
4      ///@audit-issue M using the ERC4626 directly means carrying out all the risks like.
5      /// - no slippage protections
6      /// - transferring tokens directly impact as the balance will be not tracked internally
7      /// - flash loan attacks? depends on the yield
8      /// - balance extremely low?
9  }
```

The standard ERC4626 implementation lacks protections against several attack vectors and edge cases that can harm users, particularly in DeFi contexts where adversarial behavior is common. It is never recommended to implement the standard vault.

**Recommendation:** Implement a robust vault wrapper with proper security measures instead of directly inheriting ERC4626.

**Developer Response:**

Calls to the Vault will be done through a `ERC4626Router` to avoid any slippage issues.

## 2.6 Incomplete Pause Functionality will get borrowers liquidated unfairly

**Severity:** Low

**Status:** Fixed

**Location:**

BorrowVerdeV1.sol

**Description:** The protocol implements selective pause controls for deposit, withdraw, and borrow operations but lacks comprehensive pause functionality for all critical operations including liquidations and repayments. The current pause system can create situations where users cannot protect themselves from liquidation during emergency scenarios:

```
1  bool private _pauseDeposit;
2  bool private _pauseWithdraw;
3  bool private _pauseBorrow;
4
5  modifier depositAvailable {
6      if (_pauseDeposit) revert DepositNotAvailable();
7      _;
8  }
9
10 // But liquidate() has no pause controls
11 function liquidate(address _account) external {
12     _accrue();
13     _liquidate(_account); // No pause modifier
14 }
```

**Recommendation:** Implement comprehensive pause functionality with granular controls for different emergency scenarios.

## 2.7 Missing debtCap Check in Interest Accrual

**Severity:** Low

**Status:** Closed

**Location:**

BorrowVerdeV1.sol

**Description:** The `_accrue()` function adds daily interest to `totalDebtVerde` without enforcing that the updated value stays within the configured `debtCap`.

**Recommendation:** Add a check to ensure `totalDebtVerde + interest <= debtCap` before updating the debt value.

### Developer Response:

**1. Debt cap applies to new borrowings only:** `debtCap` is enforced in `borrow()` to prevent issuance beyond the configured limit. ##### 2. Interest accrues on outstanding principal: `_accrue()` merely updates interest on existing debt; it is not an issuance function and therefore intentionally ignores `debtCap`. ##### 3. Protocol risk unchanged: Allowing interest to push `totalDebtVerde` above `debtCap` does not enable new loans or increase exposure, it simply reflects owed interest.

## 2.8 Unsafe LTV Reduction Without Active Loan Check

**Severity:** Low

**Status:** Acknowledged

**Location:**

BorrowVerdeV1.sol

**Description:** The `updateSafeLtv()` function allows the owner to reduce the `safeLtvBp` value without checking for active loans. For example, if the current safe LTV is 70% and its lowered to 40%, borrowers who were previously within safe limits may suddenly become at risk of liquidation.

**Recommendation:** Add a check to prevent lowering `safeLtvBp` while there are active loans, or enforce a grace period or staged update mechanism.

### Developer Response:

**1. safeLtvBp governs Tier1 liquidation size:** When a position hits the Tier1 band (between `safeLtvBp` and `liquidationLtvBp`), we liquidate just enough collateral to bring its LTV back down to `safeLtvBp`. Lowering `safeLtvBp` simply increases the amount seized in a Tier1 event but does not change the onchain liquidation trigger (`liquidationLtvBp`). ##### **2. 5% minimum spread enforced:** We require at least `safeLtvBp + 500bp` `liquidationLtvBp` on every update. This ensures Tier1 (partial) and Tier2+3 (full) liquidations never overlap or create edgecase gaps. ##### **3. No new insolvency risk:** Since only the volume of collateral removed in Tier1 adjusts and the hard liquidation threshold remains fixed, borrowers cannot be pushed into liquidation purely by changing `safeLtvBp`. It remains an onchain parameter for sizing partial liquidations, not for redefining when liquidations occur.

## 2.9 No Emergency Liquidation Pause During Collateral Asset Depegging Events

**Severity:** Low

**Status:** Fixed

**Location:**

BorrowVerdeV1.sol

**Description:** The protocol lacks emergency pause controls for liquidations during collateral asset depegging events. When mpETH depegs significantly from ETH, liquidations continue using potentially stale or inaccurate pricing, creating unfair liquidation scenarios for borrowers.

**Recommendation:** Implement emergency pause controls that activate during significant collateral asset depegging events to protect both borrowers and liquidators until proper pricing mechanisms are restored.

## 2.10 Liquidators can Delay Liquidations to Maximize Profits in Higher Tiers

**Severity:** Low

**Status:** Fixed

**Location:**

BorrowVerdeV1.sol

**Description:** The tiered liquidation system creates perverse incentives for liquidators to delay liquidations until positions reach higher LTV tiers for maximum profit. The protocol uses three liquidation tiers: Tier 1 (70-79% LTV) offers partial liquidation with 5% bonus, Tier 2 (80-89% LTV) offers full debt repayment with 110% collateral, and Tier 3 (90%+ LTV) offers full debt repayment with ALL collateral. This structure incentivizes liquidators to wait for Tier 3, allowing positions to deteriorate dangerously.

```
1 function _getLiquidationDeal(uint256 _debt, uint256 _collateral) internal view returns (
    uint256 _requiredVerde, uint256 _collateralOut) {
2     uint256 realLtvBp = _getLoanToValueRatioBp(_debt, _collateral);
3
4     if (realLtvBp >= 9000) {
5         // Tier 3: Best for liquidators - get ALL collateral
6         _requiredVerde = _debt;
7         _collateralOut = _collateral;
8     } else if (realLtvBp >= 8000) {
9         // Tier 2: Good for liquidators - 10% bonus
10        _requiredVerde = _debt;
11        _collateralOut = fromVERDE2Collat(_debt.mulDiv(11000, BASIS_POINTS));
12    } else if (realLtvBp >= liquidationLtvBp) {
13        // Tier 1: Least attractive - only 5% bonus, partial liquidation
14        _requiredVerde = _getLiquidationRequiredVerde(_debt, _collateral);
15        _collateralOut = fromVERDE2Collat(_requiredVerde.mulDiv(BASIS_POINTS +
            liquidationPenaltyBp, BASIS_POINTS));
16    }
17 }
```

**Recommendation:** Implement incentive structures that reward early liquidation over delayed liquidation. Consider implementing Dutch auction liquidations where liquidation rewards decrease over time, incentivizing prompt action. Alternatively, provide higher base rewards for Tier 1 liquidations to make early intervention more attractive than waiting for positions to deteriorate further.

### Developer Response:

**1. Tier update:** Weve moved Tier3 to trigger at greater then equal to 91% LTV (instead of 90%).  
##### 2. Flattened rewards above 80%: No extra bonus accrues once a position exceeds 80% LTV. All liquidations in the 80-91% range carry the same Tier2 reward. ##### 3. Optimal exit at 80%: By capping upside beyond 80%, liquidators maximally profit by acting as soon as 80% LTV is reached removing any incentive to delay.

## 2.11 Self-Liquidation Using Flash Loans to Minimize Losses

**Severity:** [Info](#)

**Status:** Acknowledged

**Location:**

BorrowVerdeV1.sol

**Description:** Borrowers facing liquidation can use flash loans to self-liquidate their positions, controlling the liquidation timing and minimizing losses compared to external liquidations. The protocol allows any address to liquidate any liquidatable position, including borrowers liquidating themselves.

**Recommendation:** Consider implementing restrictions on self-liquidation or time delays to maintain liquidation incentives.

## 2.12 Interest Accrual Results in Compounding (APY vs. APR)

**Severity:** [Info](#)

**Status:** Acknowledged

**Location:**

BorrowVerdeV1.sol

**Description:** The `_accrue()` function adds daily interest to `totalDebtVerde`, effectively compounding the debt. This causes the effective borrow cost to exceed the declared `MAX_BORROW_APR_BP` (e.g., 15% per year).

**Recommendation:** Clarify whether `borrowRatePerDay` is intended to represent APR or APY. If APR is intended, adjust the logic to avoid compounding, or document clearly that the rate is APY due to compounding behavior.



**Disclaimer:**

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts