BLOCKAPEX

SMART CONTRACT SECURITY

v 1.0

Date: 31-07-2025

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.

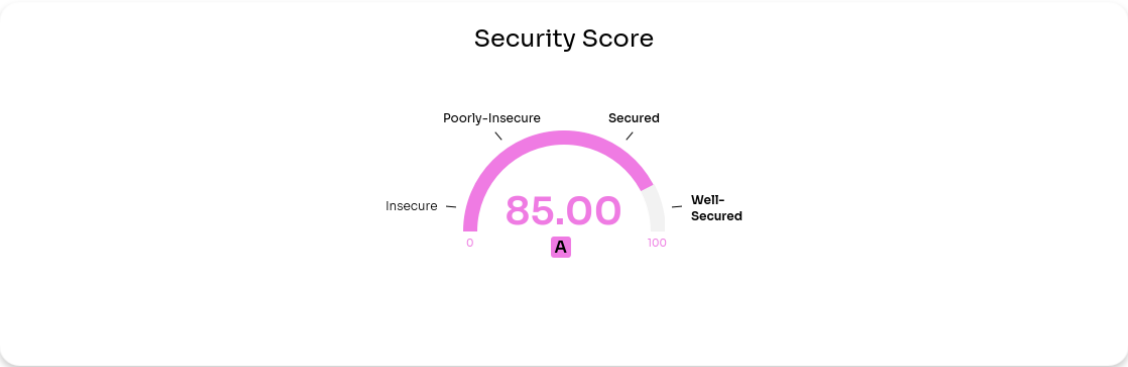
To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please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

Contents

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 85 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: Outstanding debt accrues interest daily. Interest payments contribute to protocol revenue and are tracked against a capped total debt ceiling.
- 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.

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	ACKNOWLEDGED
#3	HIGH	Flawed Daily Interest Accrual Enables Unfair Charges, Liquidation Surprises, and Interest Evasion	ACKNOWLEDGED
#4	HIGH	Incorrect mpETH Pricing Assumption Leads to Protocol Insolvency Risk	ACKNOWLEDGED
#5	HIGH	Missing _decimalsOffset() Override Leads to Potential Inflation Attack in ERC4626 Vault	FIXED
#6	HIGH	Oracle Price Limitation Enables Arbitrage and Protocol Insolvency	ACKNOWLEDGED
#7	MEDIUM	Missing Validation Oracle on latestRoundData() Exposes Protocol to Stale or Invalid Prices	FIXED
#8	MEDIUM	Direct ERC4626 Implementation Exposes Users to Multiple Vault Risks	ACKNOWLEDGED
#9	LOW	Incomplete Pause Functionality will get borrowers liquidated unfairly	FIXED
#10	LOW	Missing debtCap Check in Interest Accrual	CLOSED
#11	LOW	Unsafe LTV Reduction Without Active Loan Check	ACKNOWLEDGED
#12	LOW	No Emergency Liquidation Pause During Collateral Asset Depegging Events	FIXED

S.NO	SEVERITY	FINDINGS	STATUS
#13	LOW	Liquidators can Delay Liquidations to Maximize Profits in Higher Tiers	FIXED
#14	INFO	Self-Liquidation Using Flash Loans to Minimize Losses	ACKNOWLEDGED
#15	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:

```
// In BorrowVerdeV1.sol - assumes Verde = $1 for collateral conversions
_collat2VerdeConversionConstant = 10 ** (_collatToUsdOracle.decimals() + 18 - _verdeToken.
    decimals());

function fromCollat2VERDE(uint256 _amount) internal view returns (uint256) {
    return _amount.mulDiv(_collatPriceUsd(), _collat2VerdeConversionConstant, Math.
        Rounding.Floor);
}

// In SwapVerde.sol - assumes Verde = $1 for stablecoin swaps
_stable2VerdeConversionConstant = 10 ** (_stableToUsdOracle.decimals() + 6 - _verdeToken.
    decimals());

function fromStable2VERDE(uint256 _amount) internal view returns (uint256) {
    (, int256 price,,,) = stableToUsdOracle.latestRoundData();
    return _amount.mulDiv(_unsigned256(price), _stable2VerdeConversionConstant, Math.
        Rounding.Ceil);
}
```

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: Acknowledged

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%:

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

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.

```
// Option 1: Treasury subsidy for underwater positions
if (realLtvBp gret= 10000) {
    uint256 shortfall = _debt - fromCollat2VERDE(_collateral);
    _requiredVerde = fromCollat2VERDE(_collateral); // Pay only collateral value
    _collateralOut = _collateral;
    // Treasury covers the shortfall to incentivize liquidation
    _mintInterests(shortfall); // Mint to cover bad debt
}

// Option 2: Automated liquidation system
function _emergencyLiquidation(address _account) internal {
    // Automatically liquidate when LTV gret 100% using protocol reserves
    // Or implement a Dutch auction system for underwater collateral
}
```

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 (including LTV gret 100%). ##### 2. Safety Fund Backstop The bot is provisioned from a treasurymaintained 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 Flawed Daily Interest Accrual Enables Unfair Charges, Liquidation Surprises, and Interest Evasion

Severity: High

Status: Acknowledged

Location:

BorrowVerdeV1.sol

Description: The protocol accrues interest on outstanding debt only once per day using the _accrue() function, which updates totalDebtVerde based on the number of days elapsed (timeElapsedDays). Interest is applied uniformly to all borrowers once per call, rather than being accrued per borrower on a continuous (per-second) basis. This creates multiple issues:

1. Unfair Interest Charges: Borrowers who take out loans near the end of a day pay a full days worth of interest, even if theyve held the debt for only a few seconds. This results in disproportionate effective interest rates and unfair treatment between borrowers.
2. Sudden Liquidation Risk: All accumulated interest is applied at once, causing borrowers close to the liquidation threshold to suddenly become liquidatable. This can lead to mass liquidations in a single transaction without any advance warning to users.
3. Interest Evasion via Repayment Timing: Borrowers can avoid paying accrued interest by repaying just before the _accrue() function is called. Since interest isnt continuously updated, such borrowers effectively take out interest-free loans if they repay before the daily accrual kicks in.

Recommendation: Implement continuous interest accrual that tracks the exact timestamp of each borrowing action. Instead of daily batch processing, calculate interest on a per-second basis for each individual position.

Developer Response:

1. Deliberate daily model: Statechanging calls (borrow(), repay(), etc.) always invoke _accrue() first, updating totalDebtVerde for all loans. Readonly calls (getDebt(), view functions) compute and return the uptodate debt amount ontheflywithout mutating stateso even if _accrue() hasnt run for days, users always see their correct balance.

2. Upfront interest charge: On borrow(), we collect that days interest prorata immediately, preventing any borrowandrepaybeforeaccrue exploits.

3. Usercentric tradeoff: Our core clientele takes multiweek/month loans; subdaily precision offers negligible benefit, whereas daily batching keeps gas costs low and contract logic simple.

4. Roadmap for fixed schedules: We plan to introduce familiar amortized payment plans (e.g. weekly or monthly fixed installments) to eliminate intraday timing effects without persecond onchain computation.

2.4 Incorrect mpETH Pricing Assumption Leads to Protocol Insolvency Risk

Severity: High

Status: Acknowledged

Location:

BorrowVerdeV1.sol

Description: The protocol incorrectly assumes that mpETH (a staked ETH derivative token) has the same value as ETH, despite using mpETH as the collateral asset. The `_collatPriceUsd()` function directly fetches ETH/USD price and treats it as mpETH/USD price:

```
function _collatPriceUsd() private view returns (uint256) {
    (, int256 price,,,) = collatToUsdOracle.latestRoundData();
    /// @audit-issue H eth price assumed to be equal to collat mpETH price
    return _unsigned256(price);
}

/// @dev Convert Collateral value into USD.
function fromCollat2VERDE(uint256 _amount) internal view returns (uint256) {
    return _amount.mulDiv(_collatPriceUsd(), _collat2VerdeConversionConstant, Math.
        Rounding.Floor);
}
```

This creates a fundamental pricing error where the protocol overvalues mpETH collateral during depegging events

Recommendation: Use the existing `MPETHPriceFeed` contract or implement proper mpETH price derivation that accounts for the staking derivative's actual market value.

Developer Response:

1. Onchain true price spETHs value is determined by its builtin redemption ratio (sharestoETH) enforced by the spETH contract: anyone can unstake at that exact rate after the delay period. ##### 2. Floating market fundamental value Shortterm floating prices on DEX/CEX can deviate during panic, but holders always reclaim ETH at the onchain redemption rate. ##### 3. Protocol uses redemption rate Our collatPriceUsd() derives USD value from the onchain spETHETH share rate, not from transient secondarymarket quotes. This ensures collateral is never overvalued.

2.5 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:

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

2.6 Oracle Price Limitation Enables Arbitrage and Protocol Insolvency

Severity: High

Status: Acknowledged

Location:

MetaPoolETHOracle.sol

Description: The oracle implementation enforces an artificial 5% maximum price change per day, which is incompatible with cryptocurrency market volatility. The `_assert24HrsPriceDelta()` function prevents price updates that exceed this threshold:

```
uint256 private constant MAX_CHANGE_PER_DAY_BP = 500; // 5.00%

function _assert24HrsPriceDelta(uint256 _newMpETHPriceInETH) private view {
    uint256 _lastPrice = last24HrsPrice;

    if (_lastPrice &lt; _newMpETHPriceInETH) {
        uint256 maxDelta = _lastPrice.mulDiv(BASIS_POINTS + MAX_CHANGE_PER_DAY_BP,
            BASIS_POINTS);
        if (maxDelta &lt; _newMpETHPriceInETH) revert UpdateGreaterThanDeltaPrice(maxDelta
            , _newMpETHPriceInETH);
    } else {
        uint256 minDelta = _lastPrice.mulDiv(BASIS_POINTS - MAX_CHANGE_PER_DAY_BP,
            BASIS_POINTS);
        if (minDelta gret _newMpETHPriceInETH) revert UpdateLessThanDeltaPrice(minDelta,
            _newMpETHPriceInETH);
    }
}
```

This creates a dangerous disconnect between real market prices and protocol-reported prices, enabling sophisticated arbitrage attacks when actual market volatility exceeds 5%.

Recommendation: Remove artificial price limitations and implement proper market-responsive oracles that can handle realistic cryptocurrency volatility:

```
// Remove the problematic price delta check entirely
function updatePrice(uint256 _newMpETHPriceInETH) external onlyAuthorizedBot isAlive {
    if (_newMpETHPriceInETH &lt; SHARES_FACTOR) revert InvalidMpETHPrice();

    // Remove: _assert24HrsPriceDelta(_newMpETHPriceInETH);

    if (getElapsedTimeSinceLast24HrsPrice() gret= ONE_DAY) _updateLast24HrsPrice(
        _newMpETHPriceInETH);

    mpethPriceInETH = _newMpETHPriceInETH;
    lastPriceUpdateTimestamp = block.timestamp;

    emit NewPrice(msg.sender, _newMpETHPriceInETH);
}

// Or implement circuit breakers with emergency protocols
function emergencyPriceUpdate(uint256 _newMpETHPriceInETH) external onlyOwner {
    // Allow owner to bypass limits during extreme market conditions
    // Implement additional validation and governance requirements
}
```

Developer Response:

1. Layer2 price relay: Our oracle publishes the spETHETH rate on L2, where the native spETH contract isnt available.

2. 5% daily cap as safety guard: The 5% delta check exists to catch misfeeds or keeperbot glitches under normal market conditions, spETH/ETH will not jump more than 5% in 24hrs.

3. Offchain keeper oversight: An authorized keeper bot pulls from multiple primary data sources and only submits onchain updates when they pass internal sanity checks. If prices momentarily exceed 5%, the bot flags the update for human review rather than risk a bad feed.

2.7 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:

```
function fromStable2Verde(...) {
    (, int256 price,,,) = stableToUsdOracle.latestRoundData(); // No checks
    ...
}

function fromVerde2Stable(...) {
    (, int256 price,,,) = stableToUsdOracle.latestRoundData(); // No checks
    ...
}
```

In BorrowVerdeV1.sol:

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

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.

```
function _getFreshPrice(AggregatorV3Interface oracle) internal view returns (uint256) {
    (uint80 roundId, int256 price, , uint256 updatedAt, uint80 answeredInRound) = oracle.
        latestRoundData();

    require(price gret 0, &amp;quot;Invalid price&amp;quot;);

    require(updatedAt != 0 &amp;amp;&amp;amp; block.timestamp - updatedAt &amp;lt;= staleThreshold, &
        amp;quot;Stale price&amp;quot;);

```

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

2.8 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:

```
contract StakedVerdeV1 is ERC4626 {
    constructor(IERC20 _asset) ERC4626(_asset) ERC20(&amp;quot;Staked Meta Pool USD Stablecoin
        &amp;quot;, &amp;quot;stVERDE&amp;quot;) {}

    ///@audit-issue M using the ERC4626 directly means carrying out all the risks like.
    /// - no slippage protections
    /// - transfering tokens directly impact as the balance will be not tracked internally
    /// - flash loan attacks? depends on the yield
    /// - balance extremely low?
}
```

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.

2.9 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:

```
bool private _pauseDeposit;
bool private _pauseWithdraw;
bool private _pauseBorrow;

modifier depositAvailable {
    if (_pauseDeposit) revert DepositNotAvailable();
    _;
}

// But liquidate() has no pause controls
function liquidate(address _account) external {
    _accrue();
    _liquidate(_account); // No pause modifier
}
```

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

2.10 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.

2.11 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.

2.12 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.

2.13 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.

```
function _getLiquidationDeal(uint256 _debt, uint256 _collateral) internal view returns (
    uint256 _requiredVerde, uint256 _collateralOut) {
    uint256 realLtvBp = _getLoanToValueRatioBp(_debt, _collateral);

    if (realLtvBp gret= 9000) {
        // Tier 3: Best for liquidators - get ALL collateral
        _requiredVerde = _debt;
        _collateralOut = _collateral;
    } else if (realLtvBp &gt;= 8000) {
        // Tier 2: Good for liquidators - 10% bonus
        _requiredVerde = _debt;
        _collateralOut = fromVERDE2Collat(_debt.mulDiv(11000, BASIS_POINTS));
    } else if (realLtvBp &gt;= liquidationLtvBp) {
        // Tier 1: Least attractive - only 5% bonus, partial liquidation
        _requiredVerde = _getLiquidationRequiredVerde(_debt, _collateral);
        _collateralOut = fromVERDE2Collat(_requiredVerde.mulDiv(BASIS_POINTS +
            liquidationPenaltyBp, BASIS_POINTS));
    }
}
```

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 to91% LTV (instead of 90%). ##### 2. Flattened rewards above 80%: No extra bonus accrues once a position exceeds 80% LTV. All liquidations in the 8091% 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 reachedremoving any incentive to delay.

2.14 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.

2.15 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