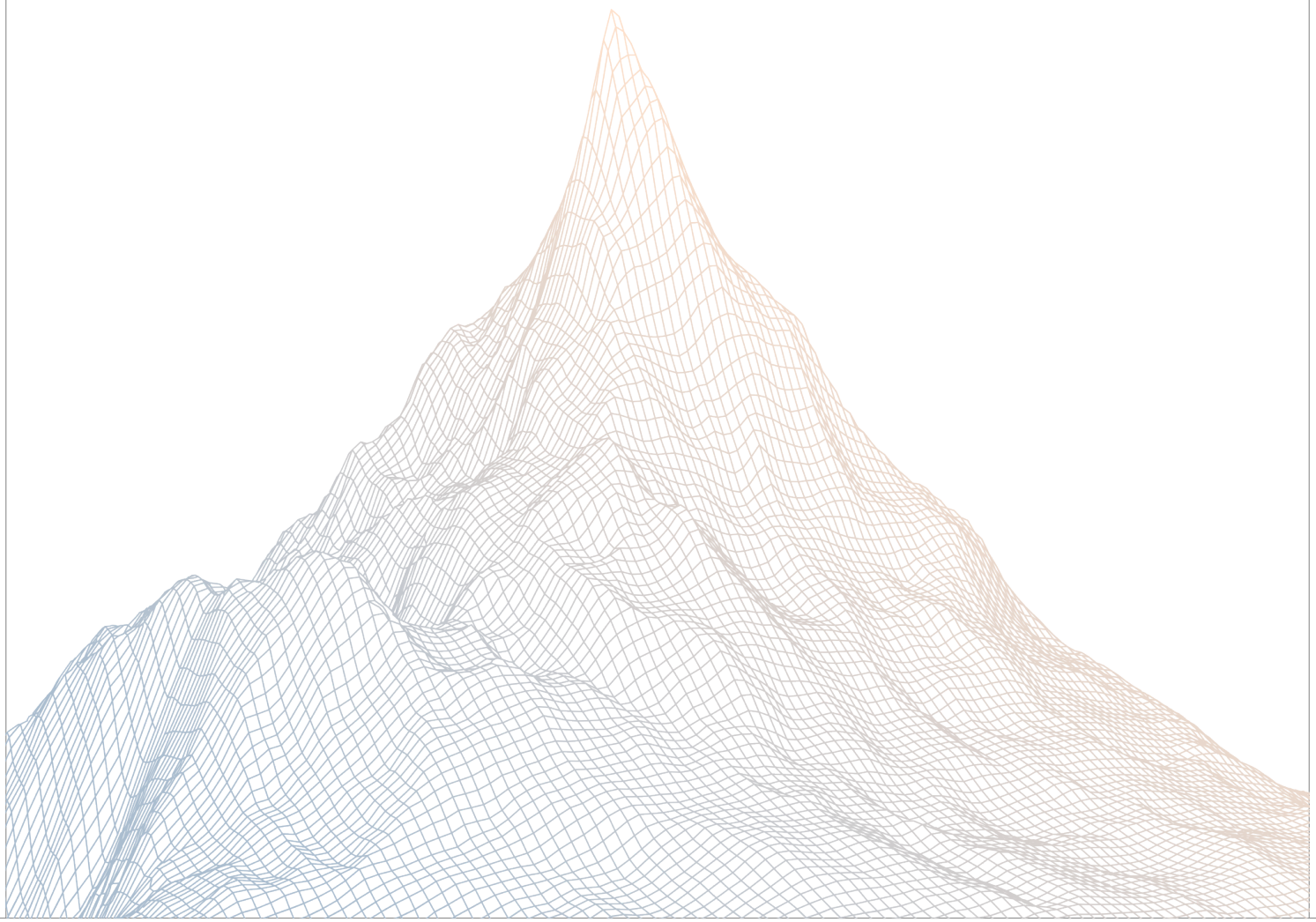
Zenith

Chateau

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 22th to August 22th, 2025

AUDITED BY: Spicymeatball
windhustler

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Chateau

Chateau is a DeFi protocol built to democratize access to institutional assets and private markets through stable-value assets and blockchain-native investment vehicles.

chUSD is a multi-collateral synthetic dollar built for multichain composability. Stake it into schUSD, an ERC-4626 vault providing on-chain access to institutional private credit yields with a focus on transparent, risk-adjusted performance.

2.2 Scope

The engagement involved a review of the following targets:

Target	chateau-usd
Repository	https://github.com/chateau-capital/chateau-usd
Commit Hash	`82e830546d00fc6e7abd2a5653dae48e9de8851d`
Files	`contracts/contracts/*`

2.3 Audit Timeline

August 22, 2025	Audit start
August 22, 2025	Audit end
September 1, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	2
Informational	1
Total Issues	**5**

3

Findings Summary

ID	Description	Status
M-1	Cross-chain minting and deployment of CHT leads to inflation beyond the maximum rate	Resolved
M-2	Missing rate limit checkpoint during configuration update	Resolved
L-1	Remove receive function from ChateauMinting	Resolved
L-2	chUSD token lacks flow rate limiter	Resolved
I-1	Zero mint delay renders CHT mint cap ineffective	Resolved

4

Findings

4.1 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Cross-chain minting and deployment of CHT leads to inflation beyond the maximum rate

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- CHT.sol#L14

Description:

The CHT token mint function enables a 10% maximum annual inflation rate. However, the contract inherits from LayerZero's OFT (Omnichain Fungible Token) and is meant to be deployed on multiple chains. The inflation check only considers the local chain's `totalSupply()`, allowing the owner to bypass the maximum inflation limit by minting tokens across multiple chains.

Each chain deployment maintains its own independent `totalSupply()` and `lastMintTimestamp` state variables. The owner can mint up to 10% of each chain's total supply per minting cycle, effectively multiplying the actual inflation rate by the number of deployed chains.

Additionally, `CHT` deployed on each chain will mint 15 billion tokens.

Recommendations:

Token minting during deployment and while calling the `CHT::mint` function should be restricted to a single chain.

Chateau: Resolved with @ff94d9c9bc...

Zenith: Verified.

[M-2] Missing rate limit checkpoint during configuration update

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- RateLimiter.sol#L31-L42

Description:

The `RateLimiter::_setRateLimits` function updates rate limit configurations without checkpointing the current state, allowing new decay parameters to be retroactively applied to existing in-flight amounts.

The issue occurs because when rate limits are updated via `_setRateLimits`, the function only updates the `limit` and `window` parameters without calling `_checkAndUpdateRateLimit` to checkpoint the current `amountInFlight` and `lastUpdated` values. This means that any new decay rate will be applied retroactively to calculate the current amount in flight, potentially allowing more tokens to be sent than originally intended.

Proof of Concept

1. Initial state: Rate limit is set with `limit = 1000` tokens and `window = 1 hour` for destination endpoint.
2. User sends 800 tokens via `chUSDOFT::_debit`, which calls `_checkAndUpdateRateLimit` and updates `amountInFlight = 800` and `lastUpdated = block.timestamp`.
3. After 30 minutes, the owner calls `chUSDOFT::setRateLimits` with a new configuration: `limit = 1000` and `window = 10 minutes`.
4. The new shorter window is applied retroactively to the 30-minute-old transaction, causing the decay calculation to treat the 800 tokens as if they were sent under the 10-minute window rule.
5. Since 30 minutes > 10 minutes window, the `_amountCanBeSent` function calculates `currentAmountInFlight = 0`, effectively resetting the rate limit.

Recommendations:

Add a checkpoint call before updating rate limit configurations to ensure the current state is properly calculated with the existing parameters before applying new ones:

```
function _setRateLimits(RateLimitConfig[] memory _rateLimitConfigs)
    internal {
    unchecked {
        for (uint256 i = 0; i < _rateLimitConfigs.length; i++) {
            RateLimit storage rl = rateLimits[_rateLimitConfigs[i].dstEid];

            // Checkpoint existing rate limit with current parameters
            _checkAndUpdateRateLimit(_rateLimitConfigs[i].dstEid, 0);

            // @dev does NOT reset the amountInFlight/lastUpdated of an
    existing rate limit
            rl.limit = _rateLimitConfigs[i].limit;
            rl.window = _rateLimitConfigs[i].window;
        }
    }
    emit RateLimitsChanged(_rateLimitConfigs);
}
```

Chateau: Resolved with @b98f1b0a22d...

Zenith: Verified.

4.2 Low Risk

A total of 2 low risk findings were identified.

[L-1] Remove receive function from ChateauMinting

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- ChateauMinting.sol#L161-L163

Description:

The `ChateauMinting` contract contains a `receive()` function that accepts Ether without any corresponding functionality to handle the received ETH. The contract only handles ERC20 tokens after WETH minting functionality was removed.

Recommendations:

Remove the unnecessary `receive()` function.

Chateau: Resolved with @b36c3a0719...

Zenith: Verified. The `receive()` function has been removed.

[L-2] chUSD token lacks flow rate limiter

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- chUSD.sol

Description:

The `chUSD` token inherits from the LayerZero OFT contract, allowing cross-chain transfers through LZ endpoints. However, unlike `chUSDOFT`, it does not implement a rate limiter feature:

```
    function _debit(
        address _from,
        uint256 _amountLD,
        uint256 _minAmountLD,
        uint32 _dstEid
    ) internal virtual override returns (uint256 amountSentLD,
    uint256 amountReceivedLD) {
>>      _checkAndUpdateRateLimit(_dstEid, _amountLD);
        return super._debit(_from, _amountLD, _minAmountLD, _dstEid);
    }
```

This omission may result in an uncontrolled outflow of tokens from the mainnet. For reference, Ethena includes limiters in its `ENA` and `USDe` OFT adapters:

- https://etherscan.io/address/0x58538e6a46e07434d7e7375bc268d3cb839c0133#code
- https://etherscan.io/address/0x5d3a1ff2b6bab83b63cd9ad0787074081a52ef34#code

Recommendations:

It is recommended to inherit from `RateLimiter` in `chUSD` to prevent excessive outflows.

Chateau: Resolved with @876767b797...

Zenith: Verified. The tokens now implement a rate limiter for secure cross-chain operations.

4.3 Informational

A total of 1 informational findings were identified.

[I-1] Zero mint delay renders CHT mint cap ineffective

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- CHT.sol#L19

Description:

The `CHT` token implements a minting cap that limits newly minted tokens to 10% of the current supply:

```
function mint(address to, uint256 amount) external onlyOwner {
    if (block.timestamp - lastMintTimestamp < MINT_WAIT_PERIOD)
    revert MintWaitPeriodInProgress();
    uint256 _maxInflationAmount = totalSupply() * MAX_INFLATION / 100;
    if (amount > _maxInflationAmount) revert MaxInflationExceeded();
    lastMintTimestamp = uint40(block.timestamp);
    _mint(to, amount);
    emit Mint(to, amount);
}
```

However, `MINT_WAIT_PERIOD` is set to 0 days, allowing multiple consecutive mints. As a result, the `maxInflationAmount` check loses its intended purpose.

Recommendations:

It is recommended to define a reasonable `MINT_WAIT_PERIOD` to enforce the cap effectively.

Chateau: Resolved with @fef9e91899...

Zenith: Verified. `MINT_WAIT_PERIOD` is set to 3 days.