hexens x Secured Finance

Security Review Report for Secured Finance

August 2025

Table of Contents

1. About Hexens

Hexens is a pioneering cybersecurity firm dedicated to establishing robust security standards for Web3 infrastructure, driving secure mass adoption through innovative protection technology and frameworks. As an industry elite experts in blockchain security, we deliver comprehensive audit solutions across specialized domains, including infrastructure security, Zero Knowledge Proof, novel cryptography, DeFi protocols, and NFTs.

Our methodology combines industry-standard security practices combined with unique methodology of two teams per audit, continuously advancing the field of Web3 security. This innovative approach has earned us recognition from industry leaders.

Since our founding in 2021, we have built an exceptional portfolio of enterprise clients, including major blockchain ecosystems and Web3 platforms.

2. Executive Summary

This report covers the security review for Secured Finance, the issuer of the USDFC stablecoin, a USD-pegged stablecoin on Filecoin Network. This review covered an incremental update introducing EIP-3009.

Our security assessment was a full review of the updates, spanning a total of 3 days.

During our review, we did not identify any major or minor severity vulnerabilities.

We did identify several code optimisations.

All of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

3. Security Review Details

▪ Review Led by

Jahyun Koo, Lead Security Researcher

▪ Scope

The analyzed resources are located on:

🔗 https://github.com/Secured-Finance/stablecoin-contracts/tree/2d1400adff8d1ab50b188c08fc4715b6ec192a93

The issues described in this report were fixed in the following commit:

🔗 https://github.com/Secured-Finance/stablecoin-contracts

📌 Commit: `c819e45531d1e3dae7670d7ed897a14818135a56`

▪ Changelog

20 August 2025	Audit start
25 August 2025	Initial report
27 August 2025	Revision received
28 August 2025	Final report

4. Severity Structure

The vulnerability severity is calculated based on two components:

1. Impact of the vulnerability
2. Probability of the vulnerability

Impact	Probability			
	Rare	Unlikely	Likely	Very likely
Low	Low	Low	Medium	Medium
Medium	Low	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

- **Severity Characteristics**

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities that are highly likely to be exploited and can lead to catastrophic outcomes, such as total loss of protocol funds, unauthorized governance control, or permanent disruption of contract functionality.

High

Vulnerabilities that are likely to be exploited and can cause significant financial losses or severe operational disruptions, such as partial fund theft or temporary asset freezing.

Medium	Vulnerabilities that may be exploited under specific conditions and result in moderate harm, such as operational disruptions or limited financial impact without direct profit to the attacker.
Low	Vulnerabilities with low exploitation likelihood or minimal impact, affecting usability or efficiency but posing no significant security risk.
Informational	Issues that do not pose an immediate security risk but are relevant to best practices, code quality, or potential optimizations.

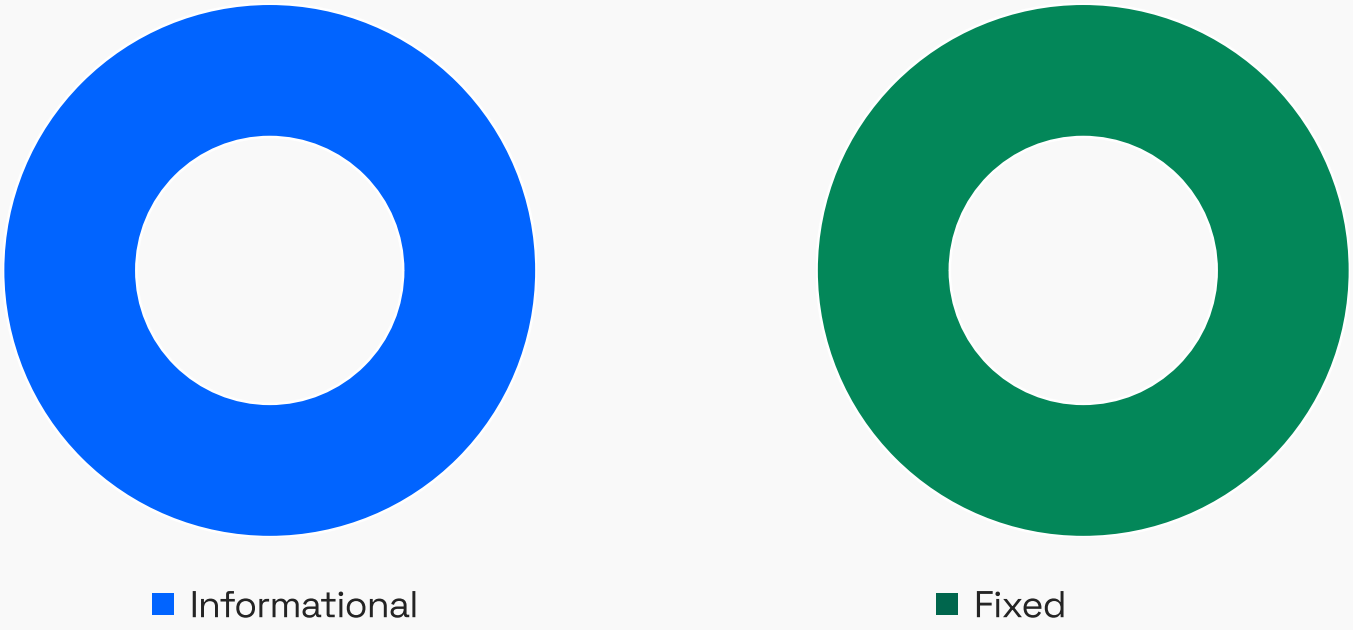
- **Issue Symbolic Codes**

Each identified and validated issue is assigned a unique symbolic code during the security research stage.

Due to the structure of the vulnerability reporting flow, some rejected issues may be missing.

5. Findings Summary

Severity	Number of findings
Critical	0
High	0
Medium	0
Low	0
Informational	2
Total:	**2**



6. Weaknesses

This section contains the list of discovered weaknesses.

SECFIN2-1 | ECDSA signature normalization (low-s/v range) not enforced (best practice)

Fixed

Severity:	Probability:	Impact:
Informational	Rare	Informational

Path:

contracts/DebtToken.sol

Description:

Several signature recoveries rely on **ecrecover** without enforcing low-s (s ≤ secp256k1n/2) and strict v-range (27 or 28). This does not bypass nonce-based replay protections but deviates from common best practices and may lead to signature malleability or interoperability inconsistencies.

Affected areas include:

- **DebtToken**: **_recover** used by **permit**, **transferWithAuthorization**, **receiveWithAuthorization**, and **cancelAuthorization**.
- **ProtocolToken**: **permit** uses **ecrecover** directly without low-s/v validation.

```
    function _recover(
        uint8 _v,
        bytes32 _r,
        bytes32 _s,
        bytes memory _typeHashAndData
    ) internal view returns (address) {
        bytes32 digest = keccak256(
            abi.encodePacked("\x19\x01", domainSeparator(),
keccak256(_typeHashAndData))
        );
        address recovered = ecrecover(digest, _v, _r, _s);
        require(recovered != address(0), "EIP712: invalid signature");
        return recovered;
    }
```

Remediation:

Prefer **ECDSA.recover** from OpenZeppelin, which enforces low-s and validates v. Alternatively, add explicit checks:

- require(uint256(s) <= SECP256K1N_HALF)
- require(v == 27 || v == 28)

SECFIN2-2 | Code clarity inconsistencies (no-op statement, missing emit)

Fixed

Severity:	Probability:	Impact:
Informational	Rare	Informational

Description:

Minor clarity issues reduce readability and can hinder observability:

- No-op statement in **BorrowerOperations.openTrove** (a standalone **vars.debtTokenFee;** has no effect).
- Missing **emit** keyword on event emissions:
  - **ActivePool.increaseDebt/decreaseDebt** call **ActivePoolDebtUpdated(debt)** without **emit**.
  - **SortedTroves._remove** calls **NodeRemoved(_id)** without **emit**, while the interface defines it as an event.

While not security-impacting, these patterns can cause confusion, compilation issues.

Remediation:

- Remove the no-op line in **BorrowerOperations.openTrove**
- Add **emit** to all event emissions (e.g., emit **ActivePoolDebtUpdated(debt)**, emit **NodeRemoved(_id))**

hexens x Secured Finance