Quantstamp

FDUSD - V2

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Stablecoin
Timeline	2025-12-01 through 2025-12-04
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• FD-121/fd-stablecoin #PR 4
Auditors	• Cameron Biniamow Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Hytham Farah Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	2 Acknowledged: 2
High severity findings	0
Medium severity findings	0
Low severity findings	1 Acknowledged: 1
Undetermined severity findings	0
Informational findings	1 Acknowledged: 1

Summary of Findings

FDUSD is an upgradeable ERC20 token with an account-freeze feature preventing token transfers, EIP-712 signatures for gasless transactions, and EIP-7598 functionality for authorized transfers, which can be enabled or disabled by the contract owner.

The audit for the `StablecoinV2` contract reveals two vulnerabilities and seven suggestions to improve code quality and adhere to best practices.

The most critical issue identified is that the `initializeV2` function is `public` and can be manipulated by any caller, as it is protected only by a `reinitializer(2)` modifier (FDUSD-1). This allows a malicious actor to front-run the initialization and alter critical state variables. It is recommended to restrict access to this function using a modifier such as `onlyOwner`, or to ensure it is called atomically via `upgradeToAndCall`. Another notable concern is the risk of front-running when disabling EIP-7598 functionality (FDUSD-2).

Auditor suggestions include implementing backward compatibility for authorization methods, fixing misleading comments about storage slot availability, adding input validations, removing unused code, and fixing a typo.

Update: All issues and suggestions have been acknowledged or fixed. FDUSD-1 and FDUSD-2 have been acknowledged but not fixed, with the client providing explanations for both. Of the suggestions, three (S1, S2, and S6) have been successfully addressed through code changes, while S3 and S5 were acknowledged without changes. S4 is mitigated as only one of the points was fixed.

ID	DESCRIPTION	SEVERITY	STATUS
FDUSD-1	`initializeV2()` **Can Be Front-Run by Arbitrary Callers**	• Low	Acknowledged
FDUSD-2	**Disabling EIP-7598 Can Be Front-Run**	• Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: `https://github.com/sgmaoben/fd-stablecoin`

Included Paths: `src`

Operational Considerations

1. Transfer authorizations are only valid in the open interval (validAfter, validBefore), meaning transactions submitted exactly at boundary timestamps will fail.
2. EIP-1271 smart contract wallet signatures are supported, requiring the `from` address to implement `isValidSignature()` if it is a contract.
3. Authorization nonces use bytes32, allowing flexible strategies, but integrators must ensure unique nonce generation to prevent collisions.
4. The `transferWithAuthorization()` function is susceptible to front-running; integrators requiring atomic execution should use `receiveWithAuthorization()` instead.
5. Signed authorizations remain valid until used or explicitly canceled, even across contract pauses or EIP-7598 disable/enable cycles.
6. The `cancelAuthorization()` function works regardless of EIP-7598-enabled state, allowing users to revoke pending authorizations even when the feature is disabled.
7. Block timestamp is used for authorization validity checks, which has ~15 second variance on Ethereum mainnet.
8. Contract upgrades via proxy must call `initializeV2()` with the correct token name to properly initialize the EIP-712 domain.
9. Frozen accounts cannot receive token transfers, including authorized transfers.
10. The `StablecoinV2` contract is upgradeable. Before upgrading the contract, the new contract should be audited to ensure a proper storage layout.

Key Actors And Their Capabilities

Owner

Inherits from OpenZeppelin's `Ownable2StepUpgradeable` with a two-step transfer mechanism.

Responsibility: Token supply management, account freezing, emergency pause, EIP-7598 feature control.

Trust Assumption: Expected to be a multisig. Has complete control over token economics and can censor any user by freezing accounts or pausing the contract.

Exclusive Functions:

StablecoinV2.sol:

- `mint()` - Mint tokens to any non-frozen address
- `enableEIP7598()` - Enable transfer authorization feature
- `disableEIP7598()` - Disable transfer authorization feature

Stablecoin.sol (Inherited):

- `mint()` - Mint tokens to the owner's address
- `burn()` - Burn tokens from the owner's address
- `freeze()` - Block all transfers/approvals for an account
- `unfreeze()` - Restore account functionality
- `pause()` - Halt all token operations globally
- `unpause()` - Resume normal operation
- `transferOwnership()` - Initiate ownership transfer

Proxy Admin

External contract controlling the transparent upgradeable proxy.

Responsibility: Upgrade contract implementation.

Trust Assumption: Separate from Owner, controlled by multisig. Will only upgrade to audited implementations.

Exclusive Functions:

- `upgrade()` - Upgrade to new implementation
- `upgradeAndCall()` - Upgrade and initialize atomically

Findings

FDUSD-1 `initializeV2()` Can Be Front-Run by Arbitrary Callers

Low | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

```
Considering on some operation related thing, we chose to use proxyAdmin.upgradeAndCall(proxy,
address(newImpl), initData) to upgrade. This should have avoided the issue mentioned
```

File(s) affected: `src/StablecoinV2.sol`

Description: `initializeV2()` is `public` and only protected by `reinitializer(2)`, allowing any externally owned account to invoke it once after the proxy upgrade. If the upgrade is performed via `upgradeTo()` (without calldata) or there is a delay before the admin submits the initializer, a malicious user can frontrun the call, choose an arbitrary `_name`, and permanently toggle `eip7598EnableFlag`. Because `__EIP712_init` persists the domain separator used for permits and authorizations, attacker-controlled initialization corrupts the signed domain and invalidates existing signatures.

Recommendation: Restrict `initializeV2()` to a trusted role (e.g., add `onlyOwner`) or ensure the upgrade always calls it atomically via `upgradeToAndCall()` so untrusted parties cannot race the initializer.

FDUSD-2 Disabling EIP-7598 Can Be Front-Run

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

```
acknowledge
```

File(s) affected: `src/StablecoinV2.sol`

Description: The contract owner can call `disableEIP7598()` to prevent calls to `transferWithAuthorization()` or `receiveWithAuthorization()`. However, a user can front-run the `disableEIP7598()` and execute some EIP-7598 function calls before the feature is disabled.

Recommendation: If it is important to prevent front-running `disableEIP7598()`, be sure to pause the contract before submitting the `disableEIP7598()` transaction.

Auditor Suggestions

S1 Add Backwards Compatibility for `receiveWithAuthorization()`

Fixed

> **Update**
> Fixed by the client in commit `464bd7b04e943467148caac727b030e998fde3ec`.

File(s) affected: `src/StablecoinV2.sol`

Description: `transferWithAuthorization()` is backwards compatible, as it supports both a EIP-712 and EIP-1271 signatures. However, `receiveWithAuthorization()` only accepts the new EIP-1271 signatures, and is not backwards compatible.

Recommendation: Add backwards compatibility by implementing the following function:

```
function receiveWithAuthorization(
    address from,
    address to,
    uint256 value,
    uint256 validAfter,
    uint256 validBefore,
    bytes32 nonce,
    uint8 v,
    bytes32 r,
    bytes32 s
)
```

S2 Storage Gap Can Be More Lenient

Fixed

> **Update**
> Fixed by the client in commit `464bd7b04e943467148caac727b030e998fde3ec`.

File(s) affected: `src/StablecoinV2.sol`

Description: The storage gap is declared as follows:

```
/**
* @dev Gap for future upgrades
* Total storage slots: 50 - 1 (mapping) = 49
*/
uint256[48] private __gap;
```

The `__gap` has 2 storage slots decremented from the full capacity of 50. However, this is unnecessary as `StablecoinV2` inherits from `Stablecoin`. So long as `Stablecoin`'s storage is not changed, the storage slots assigned for `StablecoinV2` will be iterative and will not collide with `Stablecoin`.

Recommendation: If a storage gap of 50 is desired, it can be assigned without consequences.

S3 Missing Input Validation

Acknowledged

> **Update**
> The client acknowledged the suggestion and provided the following explanation:

```
since freeze() and unfreeze is an owner only operation, it is relatively safe, we decide not to make
code changes for this version.
```

File(s) affected: `src/Stablecoin.sol`

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following instances could benefit from greater input validation:

1. `Stablecoin.freeze()` : Validate `account` is not the zero address.
2. `Stablecoin.unfreeze()` : Validate `account` is not the zero address.

Recommendation: Validate the inputs.

S4 Unused Code

Mitigated

Update

The client fixed point (2) in commit `464bd7b04e943467148caac727b030e998fde3ec` . Point (1) is partially fixed as the `ECDSAUpgradeable` and `AddressUpgradeable` imports were removed, but the libraries are still present in the contract:

```
// StablecoinV2.sol

using ECDSAUpgradeable for bytes32;
using AddressUpgradeable for address;
```

File(s) affected: `src/StablecoinV2.sol` , `src/libraries/EIP7598Constants.sol`

Description:

1. The `StablecoinV2` contract imports `ECDSAUpgradeable.sol` and `AddressUpgradeable.sol` , even though it never uses them.
2. In `EIP7598Constants` , two constants are defined but never used in the codebase:
   - `ERC1271_MAGIC_VALUE` (0×1626ba7e)
   - `EIP7598_INTERFACE_ID` (0×00000000 - placeholder)

Recommendation: Remove the unused code.

S5 Redundant Events

Acknowledged

Update

The client acknowledged the suggestion and provided the following explanation:

```
it is on purpose. If you check USDC's smart contract, it is the same.
```

File(s) affected: `src/Stablecoin.sol` , `src/StablecoinV2.sol`

Description: The `mint()` function emits the `Mint` event, which is redundant since `_mint()` already emits a `Transfer` event with the same data. Similarly, `burn()` emits the event `Burn` that contains the same data as the `Transfer` event in `_burn()` .

Recommendation: Remove the redundant events.

S6 Typo in Error Message

Fixed

Update

Fixed by the client in commit `464bd7b04e943467148caac727b030e998fde3ec` .

File(s) affected: `src/StablecoinV2.sol`

Description: In `StablecoinV2.sol` on line 32, "disalbed" should be "disabled".

```
require(eip7598EnableFlag, "EIP7598 is disalbed");
```

Recommendation: Fix the typo: "EIP7598 is disabled"

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

Repo: `https://github.com/sgmaoben/fd-stablecoin`

- `174...525 ./src/Stablecoin.sol`
- `c91...c68 ./src/StablecoinV2.sol`
- `c6e...f38 ./src/libraries/EIP7598Constants.sol`

Test Suite Results

Tests were run using `forge test`. The test suite produced 56 passing test cases.

```
Analysing contracts...
Running tests...

Ran 1 test for test/DeployStablecoin.t.sol:DeployStablecoinTest
[PASS] test_Initialize() (gas: 55339)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 229.31ms (74.38ms CPU time)

Ran 13 tests for test/EIP7598.t.sol:EIP7598Test
[PASS] testRevert_CancelAuthorization_NotAuthorizer() (gas: 23100)
[PASS] testRevert_ReceiveWithAuthorization_EIP7598Disabled() (gas: 67651)
[PASS] testRevert_TransferWithAuthorization_AlreadyUsed() (gas: 117605)
[PASS] testRevert_TransferWithAuthorization_EIP7598Disabled() (gas: 46938)
[PASS] testRevert_TransferWithAuthorization_Expired() (gas: 43155)
[PASS] testRevert_TransferWithAuthorization_Frozen() (gas: 106108)
[PASS] testRevert_TransferWithAuthorization_NotYetValid() (gas: 46909)
[PASS] testRevert_TransferWithAuthorization_Paused() (gas: 103150)
[PASS] test_CancelAuthorizatioWhenEIP7598Disabled() (gas: 55359)
[PASS] test_CancelAuthorization() (gas: 49680)
[PASS] test_ReceiveWithAuthorization() (gas: 134149)
[PASS] test_TransferWithAuthorization() (gas: 125290)
[PASS] test_TransferWithAuthorizationVRS() (gas: 124721)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 266.48ms (218.28ms CPU time)
```

```
Ran 42 tests for test/Stablecoin.t.sol:StablecoinTest
[PASS] testRevert_ExpiredPermit() (gas: 41056)
[PASS] testRevert_FreezeApprove() (gas: 72902)
[PASS] testRevert_FreezeTransfer() (gas: 73009)
[PASS] testRevert_FreezeTransferFrom() (gas: 132413)
[PASS] testRevert_InvalidBurnAmount() (gas: 24322)
[PASS] testRevert_InvalidBurnOwner() (gas: 19239)
[PASS] testRevert_InvalidFreezeOwner() (gas: 21192)
[PASS] testRevert_InvalidMintOwner() (gas: 19217)
[PASS] testRevert_InvalidNonce() (gas: 68708)
[PASS] testRevert_InvalidOwner() (gas: 21312)
[PASS] testRevert_InvalidPauseOwner() (gas: 18606)
[PASS] testRevert_InvalidPendingOwner() (gas: 49713)
[PASS] testRevert_InvalidSigner() (gas: 71716)
[PASS] testRevert_InvalidUnFreezeOwner() (gas: 21170)
[PASS] testRevert_InvalidUnpauseOwner() (gas: 18565)
[PASS] testRevert_PauseApprove() (gas: 50102)
[PASS] testRevert_PauseDecreaseAllowance() (gas: 84346)
[PASS] testRevert_PauseIncreaseAllowance() (gas: 52667)
[PASS] testRevert_PauseTransfer() (gas: 50142)
[PASS] testRevert_PauseTransferFrom() (gas: 84327)
[PASS] testRevert_ResumePauseDecreaseAllowance() (gas: 70120)
[PASS] testRevert_SignatureReplay() (gas: 111180)
[PASS] test_Approve() (gas: 56148)
[PASS] test_Burn() (gas: 31061)
[PASS] test_CancelPendingOwnership() (gas: 41205)
[PASS] test_DecreaseAllowance() (gas: 63568)
[PASS] test_EIP7598Enabled() (gas: 25720)
[PASS] test_Freeze() (gas: 84170)
[PASS] test_IncreaseAllowance() (gas: 56635)
[PASS] test_Mint() (gas: 40775)
[PASS] test_MintTo() (gas: 92678)
[PASS] test_Pause() (gas: 38166)
[PASS] test_Permit() (gas: 105488)
[PASS] test_ResumePause() (gas: 102712)
[PASS] test_ResumePauseApprove() (gas: 66023)
[PASS] test_ResumePauseIncreaseAllowance() (gas: 66378)
[PASS] test_RevertInvalidAllowance() (gas: 104415)
[PASS] test_RevertInvalidBalance() (gas: 110751)
[PASS] test_TransferFromLimitedPermit() (gas: 121841)
[PASS] test_TransferFromMaxPermit() (gas: 138554)
[PASS] test_TransferOwnership() (gas: 76643)
[PASS] test_Unfreeze() (gas: 50835)
Suite result: ok. 42 passed; 0 failed; 0 skipped; finished in 266.79ms (556.89ms CPU time)

Ran 3 test suites in 318.10ms (762.58ms CPU time): 56 tests passed, 0 failed, 0 skipped (56 total tests)
```

Code Coverage

Code coverage was generated by running `forge coverage`. While code coverage is high for the in-scope contracts, the audit team recommends improving the test suite to achieve 100% coverage in `StablecoinV2.sol`, particularly by increasing branch coverage.

File	% Lines	% Statements	% Branches	% Funcs
script/DeployStablecoin.s.sol	0.00% (0/8)	0.00% (0/9)	100.00% (0/0)	0.00% (0/2)
script/UpgradeStablecoin.s.sol	0.00% (0/8)	0.00% (0/8)	100.00% (0/0)	0.00% (0/2)
src/Stablecoin.sol	100.00% (30/30)	100.00% (20/20)	100.00% (2/2)	100.00% (10/10)
src/StablecoinV2.sol	92.68% (38/41)	93.55% (29/31)	81.25% (13/16)	91.67% (11/12)

File	% Lines	% Statements	% Branches	% Funcs
test/utils/SigUtils.sol	100.00% (8/8)	100.00% (5/5)	100.00% (0/0)	100.00% (3/3)
Total	80.00% (76/95)	73.97% (54/73)	83.33% (15/18)	82.76% (24/29)

Changelog

- 2025-12-04 - Initial report
- 2025-12-08 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.