Quantstamp

Spectrum On EVM

Executive Summary

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

Type	Token Distributor
Timeline	2025-11-17 through 2025-11-19
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• 1stdigital/fd-prism-spectrum-evm #6ec1559
Auditors	• Rabib Islam Senior Auditing Engineer • Andrei Stefan Auditing Engineer • Hamed Mohammadi Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	5 Fixed: 1 Acknowledged: 4
High severity findings	0
Medium severity findings	1 Fixed: 1
Low severity findings	3 Acknowledged: 3
Undetermined severity findings	0
Informational findings	1 Acknowledged: 1

Summary of Findings

The current audit was performed on First Digital's `SpectrumSettlement` contract.

`SpectrumSettlement` is a stateless, non-custodial contract designed for deterministic token distribution. It facilitates atomic, single-transaction settlements by leveraging the ERC-3009 `transferWithAuthorization` standard, which allows for gasless transactions on behalf of users. The contract's architecture shifts all business logic, such as client and fee package management, off-chain, relying on a trusted, whitelisted "facilitator" to submit settlement instructions as calldata. These instructions specify how to distribute funds between a primary client wallet and multiple fee recipients, supporting both percentage-based and flat-fee arrangements. Security is managed through a two-tier role-based access control system for administrative functions and reentrancy guards on settlement executions, ensuring that the contract itself holds no tokens after each transaction is complete.

We identified a few particularly important issues of note:

1. An attacker can front-run valid settlement transactions and use them to transfer tokens to the contract with no chance of recovery. This can be fixed by using `receiveWithAuthorization()` instead of `transferWithAuthorization()`.
2. The contract does not record which tokens are allowed to be used, potentially allowing agents to attempt to interact with token contracts that do not implement ERC-3009.

Documentation and tests are both of decent quality, although the tests can be improved somewhat by testing against the above front-running attack.

Fix-Review Update 2026-02-03:
The client has addressed the first issue, which was of high severity, by implementing our recommendation. The remaining issues have been addressed with reference to the the particular setup used by the client.

The client also clarified their license scheme in commit `22900f0`.

ID	DESCRIPTION	SEVERITY	STATUS
FDFI-1	**Attacker Can Front-Run `transferWithAuthorization()` Causing Tokens to Be Stuck**	Medium	Fixed
FDFI-2	**No ERC-3009 Token Compatibility Validation**	Low	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
FDFI-3	No Recovery Mechanism for Stuck Tokens	Low	Acknowledged
FDFI-4	ERC-3009 Authorization Balance Drift	Low	Acknowledged
FDFI-5	Gas-Intensive Duplicate Recipient Checking Uses Excessive Gas	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

- `contracts/SpectrumSettlement.sol`

Repo: `https://github.com/1stdigital/fd-prism-spectrum-evm`

Included Paths: `contracts/SpectrumSettlement.sol`

Key Actors And Their Capabilities

The contract supports three classes of accounts:

1. `DEFAULT_ADMIN_ROLE` aka super admin – built-in to OpenZeppelin's `AccessControl` template contract, this role grants and revokes roles from addresses.

2. `ADMIN_ROLE` aka admin – this role is capable of adding and removing facilitators.
3. **Facilitator** – this role is capable of calling the functions for settlement, using the contract to distribute to lists of recipients.

Findings

FDFI-1
Attacker Can Front-Run `transferWithAuthorization()` Causing Tokens to Be Stuck

Medium | Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `d122cb8e67de6006d11183701191f62d3d67d87e` .

Description: When a facilitator attempts to call `executeSpectrumSettlement()` or `executeSpectrumSettlementV2()`, the transaction enters a mempool where others have access to the contents of the transaction. During this period, a malicious party could take the arguments of the function call and use them to independently call `transferWithAuthorization()` on the token ahead of the facilitator's call by using a higher gas price or a MEV bundle. This would cause the tokens to be transferred to the `SpectrumSettlement` contract without calling one of the functions on the contract itself, resulting in the tokens being stuck if the corresponding ERC20 contract does not have an admin capable of recovering them.

Recommendation: Instead of `transferWithAuthorization()`, use `receiveWithAuthorization()`, as the latter function includes a check to see whether the caller is the recipient of the tokens.

FDFI-2 No ERC-3009 Token Compatibility Validation

Low | Acknowledged

> **Update**
> The client provided the following explanation:
>
> > We'll enforce the validation check on our Web2 backend by maintaining an internal allowlist of stablecoins that support EIP-3009. Since we fully control the facilitators responsible for broadcasting transactions to the smart contract (and all facilitators must be whitelisted), any unsupported tokens (e.g., those lacking EIP-3009 support) will be filtered out before reaching the contract.

Description: In `contracts/SpectrumSettlement.sol`, the contract assumes all token addresses passed to `executeSpectrumSettlement()` and `executeSpectrumSettlementV2()` properly implement ERC-3009 with `transferWithAuthorization` function. However, if a token does not implement ERC-3009 but has an empty fallback function, the call would execute without errors but no actual transfer would occur, leading to a failure where the contract believes tokens were received but the balance remains zero. Additionally, USDC implementations vary significantly across chains - native Circle-deployed USDC supports ERC-3009 on Ethereum and Base, but bridged USDC on networks like BSC and Polygon may not implement ERC-3009 at all, or may implement it with incompatible ERC-712 domain separators. This creates deployment risk where the contract may be deployed on chains where the intended token doesn't support the required authorization mechanism.

```
// SpectrumSettlement.sol:226-227
IERC3009(token).transferWithAuthorization(from, address(this), value, validAfter, validBefore, nonce, v,
r, s);
// No validation that token actually implements EIP-3009

// SpectrumSettlement.sol:279-288
IERC3009Bytes(token).transferWithAuthorization(
    from,
    address(this),
    value,
    validAfter,
    validBefore,
    nonce,
    signature
);
// If token has empty fallback, call succeeds but no transfer occurs
```

Recommendation: Consider adding on-chain token whitelist mapping to prevent facilitators from attempting settlements with incompatible tokens.

FDFI-3 No Recovery Mechanism for Stuck Tokens

Low | Acknowledged

Update

The client provided the following explanation:

> We acknowledge the auditor's observation regarding the absence of a recovery mechanism for tokens accidentally sent directly to the contract outside the intended settlement flow. However, our architecture is designed with strict controls in place that mitigate this risk operationally.
>
> Specifically, we maintain full control over the set of facilitators that are authorized to interact with the smart contract. These facilitators are whitelisted and tightly integrated into our Web2 backend, where all validation logic is enforced prior to any transaction being broadcast on-chain. This includes ensuring that only supported token types and valid flows are permitted.
>
> Furthermore, the smart contract's role in our system is intentionally minimal—functioning purely as a pass-through mechanism to route approved token transfers to end recipients. By design, no arbitrary or user-driven interaction is allowed with the contract outside the defined settlement flow, and the Web2 layer acts as the sole gateway for triggering those transfers.

Description: In `contracts/SpectrumSettlement.sol`, the contract has no function to recover tokens accidentally sent directly to the contract address outside the settlement flow. While the non-custodial design means contract balance should always be zero after successful settlements, there's no safety mechanism for recovery if tokens get stuck. This means tokens sent directly to the contract are permanently unrecoverable with no admin recovery function available.

Recommendation: Add emergency recovery function for super admin

FDFI-4 ERC-3009 Authorization Balance Drift

Low | Acknowledged

Update

The client provided the following explanation:

> We acknowledge the observation regarding potential griefing risks due to off-chain validation responsibilities. As explicitly stated in the audit, our design places the entire validation and mitigation burden on the Web2 layer, which includes both backend services and the facilita-tor implementations. This is an intentional architectural decision driven by our desire to:
>
> - Minimize on-chain complexity (and thus, on-chain validation) and unnecessary gas costs
> - Retain agility in adapting validation rules off-chain
> - Ensure tight control over facilitators, who are whitelisted and governed by internal operational protocols
>
> In this model, the Web2 backend acts as the sole gateway to the smart contract, and only pre-validated, compliant transactions are allowed to proceed. Our facilitators will have built-in safeguards to prevent unauthorized token types, malformed authorizations, or unsupported flows from ever reaching the contract.

Description: In `contracts/SpectrumSettlement.sol`, agents create off-chain EIP-3009 signed authorizations that facilitators execute on-chain at a later time. Between signature creation and execution, agents can reduce their token balance below the authorized amount, causing settlement revert. The contract provides no on-chain mitigations - no validity window enforcement, no balance verification requirements. All mitigation burden falls on off-chain Web2 services and facilitator implementations. This creates griefing opportunities where facilitators waste gas on reverting transactions, clients don't receive expected payments, and agents (maliciously or accidentally) can create many authorizations that lock up facilitator resources with reverting transactions.

Recommendation: Since contract architecture delegates all state management to Web2 services, implement comprehensive off-chain mitigations:

1. Before broadcasting any transaction the facilitator's service must query the agent's current token balance on-chain, and if it is lower than the amount specified in the settlement instruction, the transaction should not be submitted.
2. The authorization windows should be short, i.e. `validAfter` and `validBefore` should be close together.
3. The off-chain platform should track the success rate of each agent's settlements. Agents who repeatedly create authorizations that fail due to insufficient funds should face consequences, such as rate limiting, throttling, or requiring the staking of collateral.

FDFI-5 Gas-Intensive Duplicate Recipient Checking Uses Excessive Gas

Informational | Acknowledged

Update

The client provided the following explanation:

> Gas fees for transactions sent to the smart contract are paid for by our facilitators.

Description: As part of the validation process, the `_validateRecipients()` function calls `_checkDuplicateRecipient()` function for every entry in the `recipients` array. This uses a quadratic time algorithm to check whether the `recipients` array contains any duplicates whereas a linear time algorithm is feasible.

Moreover, if the array is sufficiently large, this may cause the transaction to revert due to the transaction gas limit cap being reached.

Recommendation: We recommend simply checking whether the values in the `recipients` array are monotonically increasing. This way, a linear time algorithm can be used to check whether there are any duplicates in the array, at the expense of forcing the caller to sort the array before calling.

Auditor Suggestions

S1 Ensure There Are Enough Funds to Pay for All the Fees

Acknowledged

Update

The client provided the following explanation:

> Our Web2 backend can validate set up of fee structure(s) that route to fee recipients, and aspects of the settlement flow (during the process of submission involving our web2 rails). This can ensure situations such as the total fees not exceeding the available funds for the call, for example.

File(s) affected: `SpectrumSettlement.sol`

Description: The protocol allows fee recipients to receive the fees either as a percentage of the total funds in the call or as a flat fee. The `_validateInstruction()` function ensures that the sum of the percentage fees does not exceed 100%. However, it does not check the sum of flat fees. This is despite the fact that the `_validateRecipients()` function calculates both the sum of flat fees and basis points fees.

Recommendation: Consider adding validation to ensure the total fees do not exceed the available funds for the call. Returning early when funds are insufficient can also help reduce gas costs.

S2 XOR Fee Type Validation May Be Overly Strict

Acknowledged

Update

The client provided the following explanation:

> Acknowledged that the contract enforces exactly one fee type per recipient using XOR logic. However, on the web2 system side, there are opportunities to have different fee split setups per client such that complex fee structures (i.e. – “2% + $10 flat fee") can be accommodated.

Description: In `contracts/SpectrumSettlement.sol`, the contract enforces exactly one fee type per recipient using XOR logic. This prevents recipients from having both percentage AND flat fees simultaneously, which might be a legitimate use case for complex fee structures (e.g., "2% + $10 flat fee"). The current implementation limits flexibility for fee configurations that combine both percentage-based and fixed-amount components.

Recommendation: Consider allowing combined fee types if legitimate use case exists

S3 Consider Applying `nonReentrant` as the First Modifier

Acknowledged

Update

The client provided the following explanation:

> Will consider.

File(s) affected: `SpectrumSettlement.sol`

Description: The two functions `executeSpectrumSettlement` and `executeSpectrumSettlementV2()` take advantage of the OpenZeppelin `ReentrancyGuard.nonReentrant` modifier to battle possible reentrancy issues. However, these modifiers appear after the `onlyFacilitator` modifier. In Solidity, chained modifiers are applied left to right, meaning the first modifier in the chain will also be applied to

other modifiers that come after it. Therefore, placing `nonReentrant` as the first in the chain of function modifiers can supercharge its defensive abilities by protecting other modifiers from reentrancy as well.

Recommendation: Consider applying `nonReentrant` as the first modifier.

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/1stdigital/fd-prism-spectrum-evm`

- `a35...c0d ./contracts/SpectrumSettlement.sol`

Test Suite Results

Test results were obtained using `npx hardhat coverage`.

Update: `SpectrumSettler` was removed from the repository along with its tests.

```
SpectrumSettlement
  Deployment
    ✔ Should grant DEFAULT_ADMIN_ROLE to deployer
    ✔ Should grant FACILITATOR_MANAGER_ROLE to deployer
    ✔ Should have correct constants
  Facilitator Management
    addFacilitator
      ✔ Should allow FACILITATOR_MANAGER_ROLE to add facilitator
      ✔ Should revert if non-admin tries to add facilitator (43ms)
      ✔ Should revert if adding zero address
      ✔ Should revert if facilitator already exists
    removeFacilitator
      ✔ Should allow FACILITATOR_MANAGER_ROLE to remove facilitator
      ✔ Should revert if non-admin tries to remove facilitator
      ✔ Should revert if facilitator doesn't exist
  Instruction Validation
    ✔ Should revert if clientWalletAddress is zero
    ✔ Should revert if clientId is empty
    ✔ Should revert if clientId is too long
    ✔ Should revert if packageTier is 0
```

```
    ✔ Should revert if packageTier > 99
    ✔ Should revert if attestationHash is empty
    ✔ Should revert if attestationHash is too long
    ✔ Should revert if recipients array is empty
    ✔ Should revert if recipients array exceeds MAX_RECIPIENTS (391ms)
  Recipient Validation
    ✔ Should revert if recipient has zero address
    ✔ Should revert if recipient has both basisPoints and flatAmount
    ✔ Should revert if recipient has neither basisPoints nor flatAmount
    ✔ Should revert if duplicate recipients
    ✔ Should revert if total basis points exceed 100%
  Settlement Execution - RSV Variant
    ✔ Should revert if caller is not facilitator
    ✔ Should execute settlement with percentage fees successfully (65ms)
    ✔ Should execute settlement with flat fees successfully
    ✔ Should execute settlement with mixed fees (percentage + flat)
    ✔ Should handle 100% fee distribution (nothing to client)
    ✔ Should revert if flat fees exceed total amount
  Settlement Execution - Bytes Variant
    ✔ Should execute settlement with bytes signature successfully
  View Functions
    ✔ Should check facilitator manager role correctly
    ✔ Should check super admin role correctly
    ✔ Should check facilitator whitelist correctly
    ✔ Should simulate distribution with percentage fees
    ✔ Should simulate distribution with flat fees
    ✔ Should simulate distribution with mixed fees
  Security
    ✔ Should prevent non-facilitator from executing settlement
    ✔ Should have reentrancy protection on settlement functions
    ✔ Should handle different client wallet addresses per settlement (42ms)
  Gas Benchmarking
⛽ Gas used for 2 recipients: 155551
    ✔ Should benchmark gas for settlement with 2 recipients


  41 passing (3s)
```

Code Coverage

`SpectrumSettlement` has 90.32% branch coverage. It can still be increased somewhat.

Update: `SpectrumSettlement` branch coverage has diminished slightly, down to 89.06%.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	89.06	100	100	
SpectrumSettlement.sol	100	89.06	100	100	
contracts/interfaces/	100	100	100	100	
IERC3009.sol	100	100	100	100	
IERC3009Bytes.sol	100	100	100	100	
contracts/mocks/	51.85	28.57	50	50	
MockERC20.sol	0	100	0	0	14,18,22,26
MockERC3009.sol	58.33	28.57	75	55	... 76,77,78,79
MockERC3009Bytes.sol	58.33	28.57	75	56.25	... 66,67,68,69

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	84.88	70.65	77.78	84.38	

Changelog

- 2025-11-19 - Initial report
- 2026-01-30 - 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 1100 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, Solana, Polygon, TON, Cardano, Binance Smart Chain, Avalanche, Arbitrum, Flow
- DeFi: Lido, Ethena, Compound, Curve, Venus, PancakeSwap, Polymarket
- Infra/Staking: TrustWallet, Alchemy, Liquid Collective, Kiln, Galxe, API3, ssv.network, Luganodes
- Immersive: Virtuals Protocol, Wayfinder, OpenSea, Square Enix, Parallel, Camp, Decentraland
- Institutional: Visa, Circle, Revolut, Anthropic, OpenAI, Canton, Republicc, Arkham, Sequoia

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.