SentaTrust

SentaTrust Security Audit Report

Know for sure.

Senta Sample Vault

CONTRACT REVIEWED

SentaSampleVault.sol

Audit ID: SENTA-2026-990031

06 Nov 2026

Table of Contents

Disclaimer	2
Executive Summary	5
Firm Overview	5
Methodology	6
Audit Scope	6
Audit Assumptions & Trust Model	7
Findings Summary	8
Detailed Findings	9
Governance & Trust Observations	12
Privilege Review	13
Security Posture & Risk Analysis	14
Remediation Roadmap	16
Recommendations	16
Conclusion	17
Technical Appendix	18
Certification Sign-Off	19

Scope of Assessment

This report is a full SentaShield audit under the SentaTrust Standard. It covers the smart contracts listed in the Audit Scope section, as submitted, within the limits of the Essential package. It comes with a certificate, the Certification Seal and a public certificate page. What the audit does not cover is set out in the Audit Scope section and the Disclaimer.

Disclaimer

1. Point-in-Time Assessment

This security audit report represents a point-in-time analysis of the smart contract source code and documentation provided as of the audit date specified in this report. The audit findings are based exclusively on the code version specified in the Audit Scope section and do not extend to:

- Subsequent code changes, updates, or modifications after the audit date
- On-chain behavior of deployed contracts (if deployment occurs post-audit)
- Economic, game-theoretic, or governance risks beyond code-level vulnerabilities
- Front-end applications, user interfaces, or off-chain infrastructure
- Network-level attacks, including MEV exploitation, censorship, or denial of service
- Regulatory compliance, legal standing, or tax implications

Any changes to the audited code—including compiler upgrades, dependency updates, or protocol modifications—may invalidate the findings. A new audit is strongly recommended following significant code changes.

2. No Warranty or Guarantee

SentaTrust and its representatives make **no warranties, express or implied**, regarding the security, functionality, correctness, or fitness for any purpose of the audited code. This report is provided strictly on an "as is" and "as available" basis.

This audit does not guarantee:

- The absence of vulnerabilities, bugs, or exploitable conditions
- That the audited code will function as intended under all conditions

- Protection against all attack vectors, including zero-day exploits
- That no security issues will be discovered in the future

Smart contract security is an evolving field. The absence of identified vulnerabilities in this report does not constitute proof of security.

3. No Endorsement or Investment Advice

This report is a technical security assessment and **does not constitute**:

- An endorsement or recommendation of the audited project or protocol
- Investment advice, financial advice, or trading recommendations
- Legal, regulatory, or compliance guidance
- A certification, approval, or validation of the project's business model, economic design, or long-term viability

Users and investors must conduct their own due diligence. The presence of an audit—whether favorable or critical—should not be interpreted as a signal to buy, sell, or hold any digital asset or participate in any protocol.

SentaTrust has no financial interest in the audited project beyond the professional services fee for conducting this audit.

4. External Dependencies and Trust Assumptions

This audit assumes the correctness and security of all external dependencies, including but not limited to:

- Third-party libraries and imported smart contracts (e.g., OpenZeppelin, Solmate)
- External oracles, price feeds, and off-chain data sources
- Underlying blockchain protocols and consensus mechanisms
- Compiler implementations and development toolchains

External code is NOT audited unless explicitly stated in the Audit Scope section. Vulnerabilities in external dependencies may compromise the security of the audited system, regardless of the audit findings.

The "Audit Assumptions & Trust Model" section of this report explicitly documents all trust boundaries and out-of-scope dependencies identified during the review.

5. Limitation of Liability

To the maximum extent permitted by applicable law, SentaTrust, its officers, employees, contractors, and auditors **shall not be liable** for any direct, indirect, incidental, consequential, special, or punitive damages arising from:

- The use, reliance upon, or distribution of this report
- The deployment, operation, or exploitation of the audited code
- Any security breach, vulnerability, or loss of funds in the audited system
- Any errors, omissions, or inaccuracies in this report

This limitation applies regardless of the form of action (contract, tort, negligence, strict liability, or otherwise), even if SentaTrust has been advised of the possibility of such damages.

Nothing in this disclaimer shall exclude or limit liability for:

- Death or personal injury caused by negligence
- Fraud or fraudulent misrepresentation
- Any liability that cannot be excluded or limited under applicable law

6. No Third-Party Reliance

This report is prepared exclusively for the client identified in the Audit Scope section ("the Client"). **No third party** (including but not limited to end users, investors, token holders, protocol participants, or the general public) may rely on this report for any purpose without the express prior written consent of SentaTrust.

Third parties are **explicitly advised** that:

- This report was not prepared with their interests or circumstances in mind
- SentaTrust owes no duty of care to any party other than the Client
- Reliance on this report by third parties is at their own risk

The Client may publish or distribute this report, but such publication does not create any liability, duty, or obligation on the part of SentaTrust to third parties who access the report.

7. Report Usage and Publication Rights

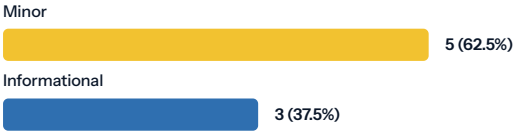
- **Client Rights:** The Client is granted a non-exclusive, non-transferable license to use this report for internal purposes, including public disclosure, regulatory compliance, and stakeholder communication.
- **Publication Permission:** SentaTrust permits the Client to publish this report in its entirety on public platforms (websites, GitHub repositories, audit registries, etc.) for transparency and community verification purposes.
- **Modification Prohibition:** The Client may **not** modify, excerpt, or misrepresent the contents of this report. Any publication must include this Disclaimer in full. Selective quotation or partial disclosure of findings without complete context is prohibited.
- **SentaTrust Rights:** SentaTrust reserves the right to reference this audit in its own marketing materials, case studies, and public portfolio, and to publish anonymized or aggregated findings for research and educational purposes.

Report Date: 06 Nov 2026
Report Version: Version 1.0

Executive Summary

SentaScore: A⁻ · 90/100 · Governance risk: High

Scored Code Defects



SentaScore™ Assessment



Risk Level: 0 Critical, 0 Major, 5 Minor, 3 Informational

What the SentaScore means. The SentaScore rates the security of the code from 0 to 100, based on the severity of confirmed defects, and expresses it as a letter: A+ (97–100), A (90–96), B (80–89), C (70–79), D (60–69), F (below 60). It describes the code at the time of the review. It is not a guarantee and not an investment signal.

What the "–" means. A minus after the letter means the code scores well, but the review found a critical governance characteristic, for example an owner who can move users' funds. The minus does not change the score. It tells the reader to weigh who controls the contract alongside the grade.

Governance risk is rated separately and never lowers the SentaScore:

- **Low:** nothing that requires meaningful trust in the operators.
- **Elevated:** characteristics such as upgradeability or privileged roles that the reader should know about.
- **High:** at least one critical characteristic, such as admin custody of user funds, that requires significant trust in the operators.

Details are in *Governance & Trust Observations*.

Senta Sample Vault received a SentaScore of A- (90/100). The review confirmed 8 code defects: 0 critical, 0 major, 5 minor and 3 informational. 3 governance and trust characteristics were reviewed and are reported separately; they do not affect the score. The code shows a strong security posture.

Severity	Code defects count towards the score	Governance & trust reviewed, not scored
▲ Critical	0	1
◆ Major	0	1
• Minor	5	1
○ Informational	3	–
Total	**8**	3

The audit reviewed 11 items: 8 scored code defects and 3 items reviewed and not scored. All scored defects should be reviewed and addressed as part of ongoing security improvements.

Firm Overview

SentaTrust is an independent verification company based in Dubai. We put the technology running your world to the test, and publish results others can check.

Our smart contract audits examine what the code does, who controls it, and what it relies on to stay safe. Every finding is graded against one documented standard, the SentaTrust Standard, so the same issue always receives the same grade.

The analysis is carried out by the SentaLabs Engine, developed by SentaLabs, our research and development division. The engine applies the standard the same way to every contract, so results are consistent and can be reproduced.

Our audits serve project teams preparing a launch, exchanges and platforms deciding on a listing, and investors checking a contract before they commit. Findings are assessed not only for what can be exploited today, but also for what changes in upgrades, administration and operation could mean for users.

SentaTrust is independent of the projects it assesses. We do not develop, deploy, promote or invest in them.

Every SentaShield audit is checked and approved by SentaTrust before it is issued. Every certificate has a public page: anyone can open it, see the grade and the findings, and confirm with the report's integrity hash that nothing has been changed. Results are published whatever the grade.

Methodology

Every audit follows the same method, applied the same way to every contract.

Code analysis

The SentaLabs Engine reads the Solidity source of every contract in scope and checks it for known classes of weakness and unsafe patterns. It also compares the contract's structure with patterns seen in documented incidents. Each confirmed defect is recorded with its location in the code.

Logic and access control

The engine follows how data and control move through the contract: state changes, access checks and edge cases.

Who controls the contract

The review identifies who holds power over the contract (owners, administrators, upgrade and pause rights) and what the contract relies on to stay safe. These governance and trust observations are reported separately and never lower the SentaScore.

Grading

Every finding is graded against the SentaTrust Standard as Critical, Major, Minor or Informational. The SentaScore (0 to 100, expressed as a letter) reflects confirmed code defects only.

Approval

Every audit is checked and approved by SentaTrust before the report and certificate are issued.

This method gives an evidence-based picture of the code as submitted, at the time of the review. No review can prove that code is free of every weakness: a review that finds nothing is not evidence that a contract is safe.

Audit Scope

This audit evaluated the security and correctness of the smart contracts and protocol components provided by the project at the time of assessment. The review focused on identifying vulnerabilities, logical errors, architectural weaknesses, and trust assumptions that could materially impact the safety, reliability, or intended behavior of the system.

Engagement Identity

- **Project:** Senta Sample Vault
- **Audit ID:** SENTA-2026-990031
- **Audit Package:** Essential
- **Network:** Ethereum

Declared Root Contracts

- SentaSampleVault.sol

In-Scope Components

- Smart contract logic and state transitions
- Access control mechanisms and privilege boundaries
- Upgradeability patterns and administrative controls
- Protocol architecture and trust assumptions
- Error handling, validation logic, and failure modes

- Interactions between in-scope contracts

Out-of-Scope Components

- Frontend interfaces, user experience, and off-chain infrastructure
- Operational security, key management, and deployment practices
- Third-party services, oracles, and external systems
- Economic modeling, incentive design, and market dynamics
- Network-level attacks or consensus-layer risks

Explicit Exclusions or Assumptions

- The audit was conducted under the assumption that all reviewed contracts represent the intended production deployment unless otherwise stated.
- This audit represents a point-in-time assessment of the provided codebase and does not guarantee the absence of vulnerabilities beyond the scope described above.

Audit Assumptions & Trust Model

System-Derived Dependency & Trust Surface

This audit assessed the smart-contract sources provided for SENTA-2026-990031 (Senta Sample Vault) on Ethereum, examining observable on-chain behavior, access-control configuration, upgrade-mechanism indicators, and dependency signals present within the single-file code snapshot submitted for review.

External library reliance indicators were detected, consistent with the use of a recognized external code framework (OpenZeppelin) within the audited source. No upgradeability pattern indicators were detected in the audited source snapshot; this observation is limited to the indicators checked by this analysis and does not rule out upgradeability introduced through deployment configuration, proxy infrastructure external to the snapshot, or subsequent code changes. No off-chain dependency indicators were detected in the audited source snapshot, again limited to the scope of indicators examined.

Privileged role and administrative control indicators were detected, including an ownership-based access-control pattern and a signal consistent with an emergency pause mechanism. Where such indicators are present, the trust model for the audited system depends materially on the integrity of role holders, the security of the credentials or infrastructure controlling those roles, and the governance processes that authorize and constrain the exercise of privileged functions, including any functions that may affect custody or control of user-deposited assets.

Conditional Disclosures

The following disclosures apply only where the corresponding dependencies or trust boundaries are present. Where a category is listed as present but no itemized disclosures are included, this indicates that no explicit, contract-specific disclosure items were generated for that category in this run.

Finding-Evidenced Off-Chain Disclosures

The following items are derived from audit findings and indicate that the audited system may rely on off-chain or external data dependencies. These items are disclosures derived from reported findings and do not modify the manifest-backed trust surface classification.

Finding-Evidenced Off-Chain Data Dependency

The audit assumes that any referenced off-chain data sources are available, correct, and operated as intended. External data dependencies are outside the scope of on-chain code review.

Upgradeability & Governance Trust

Where privileged roles, administrative controls, or upgrade mechanisms are indicated, the security and expected behavior of the system depend on secure key management, correct configuration of the authorities holding such roles, and disciplined governance and operational processes for proposing, reviewing, and executing privileged actions.

Where an emergency control or pause-type mechanism is indicated, reliance is placed on the responsible party exercising that control in a timely and appropriate manner consistent with its intended protective purpose, and on the absence of unauthorized or adversarial use of that same mechanism.

Where privileged functions may affect custody or disposition of assets held by the system, the trust model additionally depends on the custodial discipline, authorization procedures, and oversight applied to the holders of those privileges.

Economic & Game-Theoretic Assumptions

Economic and game-theoretic assumptions are treated as contextual requirements and are not proven by mechanical code correctness alone; this audit does not model adversarial market behavior, incentive manipulation, or economic attack strategies beyond constraints explicitly stated within scope.

Environment & Deployment Assumptions

This audit assumes that deployment parameters and the execution environment, including network semantics and configuration, are consistent with the intended operational context; deployment-time changes or misconfiguration may alter the effective trust model described herein.

Where the audited logic relies on block-level timestamp values as a proxy for real-world time, the trust model assumes that such reliance remains within tolerances appropriate to the system's intended use and that timestamp manipulation, within the bounds permitted by the underlying network's consensus rules, does not materially undermine the system's intended guarantees.

Standard Assumptions

This section summarizes trust boundaries and assumptions relevant to interpreting the audit results. Conclusions are limited to the provided code snapshot and the indicators observable within it, and do not extend to unprovided infrastructure, operational controls, external systems, or deployment-time configuration not reflected in the submitted source.

The scope of this assessment is bounded by the materials made available for review; limitations inherent to static analysis of a point-in-time code snapshot should be considered when applying these assumptions to the system's ongoing or future operation.

Findings Summary

SentaScore: A⁻ · **90/100** · Governance risk: High

Severity	Code defects count towards the score	Governance & trust reviewed, not scored
▲ Critical	0	1
◆ Major	0	1
• Minor	5	1
○ Informational	3	–
Total	**8**	3

Of 11 reviewed items, 8 are confirmed code defects that determine the SentaScore (see *Detailed Findings*). 3 are governance and trust characteristics, reviewed and reported separately (see *Governance & Trust Observations*); they do not lower the score.

Severity reflects impact and likelihood in the reviewed context and helps prioritise fixes. It is not a prediction that a weakness will be exploited.

Detailed Findings

Confirmed Findings

Established defects in this code. These drive the SentaScore.

Minor Findings (5)

ST-001 — Address is not checked against the zero address

• MINOR

DESCRIPTION

This finding identifies code that assigns an address parameter without a zero-address check at `SentaSampleVault.sol:39`. A zero address can disable functionality or lock privileged control if set by mistake.

LOCATION

`SentaSampleVault.sol:39`

RECOMMENDATION

Validate address parameters are non-zero before assigning or using them.

CODE SNIPPET

```
35:     event RewardsDistributed(uint256 amount, uint256 recipients);
  36:     event FeeRecipientUpdated(address oldRecipient, address newRecipient);
  37:     event TokensRescued(address token, address to, uint256 amount);
  38:
  39:     constructor(IERC20 _asset, IERC20 _rewardToken, address _feeRecipient)
  40:         ERC20("Senta Sample Vault Share", "sSVS")
  41:         Ownable(msg.sender)
  42:     {
  43:         asset = _asset;
```

ST-002 — Exact equality check on a value others can influence

• MINOR

DESCRIPTION

This finding identifies code that uses a strict equality (`==`) on a value an attacker can influence at `SentaSampleVault.sol:54`. Exact-match checks on balances, timestamps, or amounts can be bypassed or bricked by nudging the value off the exact target.

LOCATION

`SentaSampleVault.sol:54`

RECOMMENDATION

Avoid strict equality (==) on balances/timestamps/amounts an attacker can influence; use >= / <= threshold comparisons.

CODE SNIPPET

```
52:     }
  53:
  54:     function convertToShares(uint256 assets) public view returns (uint256) {
  55:         uint256 supply = totalSupply();
  56:         if (supply == 0) {
  57:             return assets;
  58:         }
  59:         return (assets * supply) / totalAssets();
  60:     }
```

ST-003 — Division before multiplication loses precision

• MINOR

DESCRIPTION

This finding identifies code that divides before multiplying in an arithmetic expression at `SentaSampleVault.sol:90`. Integer division truncates first, so the result loses precision relative to multiplying first.

LOCATION

`SentaSampleVault.sol:90`

RELATED FINDINGS

Also flagged on this line: ST-004.

RECOMMENDATION

Reorder arithmetic to multiply before dividing so integer truncation does not discard precision; otherwise scale by a fixed-point factor first.

CODE SNIPPET

```
 92:         require(balanceOf(msg.sender) >= shares, "Insufficient shares");
  93:         require(block.timestamp >= lastDepositTime[msg.sender] + lockPeriod, "Still locked");
  94:
  95:         uint256 assets = convertToAssets(shares);
  96:         uint256 fee = assets / 10000 * withdrawalFeeBps;
  97:         assetsOut = assets - fee;
  98:
  99:         _burn(msg.sender, shares);
 100:
```

ST-004 — Logic depends on the block timestamp

• MINOR

DESCRIPTION

This finding identifies code that relies on `block.timestamp` for a comparison in time-based logic at `SentaSampleVault.sol:90`. Validators have limited latitude over the timestamp, so thresholds gated on it can be nudged within that tolerance.

LOCATION

`SentaSampleVault.sol:90`

RELATED FINDINGS

Also flagged on this line: ST-003.

RECOMMENDATION

Do not use block.timestamp for thresholds an attacker could game within validator tolerance; widen the window or anchor to block numbers.

CODE SNIPPET

```
 89:     /// @notice Withdrawals stay open while the vault is paused: pausing only stops new deposits.
  90:     function withdraw(uint256 shares) external nonReentrant returns (uint256 assetsOut) {
  91:         require(shares > 0, "Zero shares");
  92:         require(balanceOf(msg.sender) >= shares, "Insufficient shares");
  93:         require(block.timestamp >= lastDepositTime[msg.sender] + lockPeriod, "Still locked");
  94:
  95:         uint256 assets = convertToAssets(shares);
  96:         uint256 fee = assets / 10000 * withdrawalFeeBps;
  97:         assetsOut = assets - fee;
```

ST-005 — Local variable used before a value is set

• MINOR

DESCRIPTION

This finding identifies code that uses a local variable before assigning it at `SentaSampleVault.sol:118`. The default value can be read in place of the intended one, producing incorrect results.

LOCATION

`SentaSampleVault.sol:118`

RECOMMENDATION

Initialize all local variables before use.

CODE SNIPPET

```
114:         require(supply > 0, "No shares");
  115:
  116:         rewardToken.safeTransferFrom(msg.sender, address(this), amount);
  117:
  118:         uint256 paid;
  119:         for (uint256 i = 0; i < depositors.length; i++) {
  120:             address user = depositors[i];
  121:             uint256 share = balanceOf(user) * amount / supply;
  122:             if (share > 0) {
```

Informational Findings (3)

ST-006 — Inconsistent compiler versions across files

○ INFORMATIONAL

DESCRIPTION

This finding identifies code that declares more than one, or inconsistent, version pragma across the sources at `SentaSampleVault.sol:2`. Inconsistent pragmas can lead to sources compiling under different compiler behaviors.

LOCATION

`SentaSampleVault.sol:2`

RELATED FINDINGS

Also flagged on this line: ST-007.

RECOMMENDATION

Use a single, consistent pragma across all files; avoid conflicting version directives.

CODE SNIPPET

No code snippet available for this finding.

ST-007 — Compiler version is too broad or has known issues

○ INFORMATIONAL

DESCRIPTION

This finding identifies code that uses a Solidity version pragma that is overly broad or ties to a version with known issues at `SentaSampleVault.sol:2`. Compiler behavior and bug fixes differ across versions, so the pragma should be narrowed to a vetted release.

LOCATION

`SentaSampleVault.sol:2`

RELATED FINDINGS

Also flagged on this line: ST-006.

RECOMMENDATION

Pin one audited, recent compiler version; avoid floating ranges and known-buggy releases.

CODE SNIPPET

No code snippet available for this finding.

ST-008 — Address in an event cannot be searched

○ INFORMATIONAL

DESCRIPTION

This finding identifies code that emits an event whose address parameter is not indexed at `SentaSampleVault.sol:36`. Un-indexed addresses cannot be filtered efficiently off-chain, reducing observability.

LOCATION

`SentaSampleVault.sol:36`

RECOMMENDATION

Index address parameters in events so off-chain consumers can filter by them.

CODE SNIPPET

```
32:
  33:     event Deposited(address indexed user, uint256 assets, uint256 shares);
  34:     event Withdrawn(address indexed user, uint256 assets, uint256 shares, uint256 fee);
  35:     event RewardsDistributed(uint256 amount, uint256 recipients);
  36:     event FeeRecipientUpdated(address oldRecipient, address newRecipient);
  37:     event TokensRescued(address token, address to, uint256 amount);
  38:
  39:     constructor(IERC20 _asset, IERC20 _rewardToken, address _feeRecipient)
  40:         ERC20("Senta Sample Vault Share", "sSVS")
```

Governance & Trust Observations

The characteristics below were **reviewed and are reported as observations**, not scored as defects. Whether each is a concern depends on operational context the audit cannot determine (it may be by design). They did **not** deduct from the SentaScore; each is shown with its true severity, the specific conditions under which it is acceptable, and the trust it requires.

Admin / Privileged Control

Single Admin Control with Emergency Powers BY DESIGN

▲ CRITICAL Reviewed — not scored

DESCRIPTION

Contract has a single admin/owner with emergency powers (pause, withdraw, upgrade, sweep) but no timelock or multisig protection. This creates a single point of failure and requires trust in the admin.

Acceptable Conditions

- Trust assumptions are explicitly documented
- Risk mitigation measures are implemented

Trust Assumptions

- Trust that admin will not abuse emergency powers
- Trust that admin key will not be compromised
- Trust that admin will act in users' best interests

Global Pause Functionality BY DESIGN

◆ MAJOR Reviewed — not scored

DESCRIPTION

Contract has global pause functionality that can halt all operations. This requires trust that pause will only be used for legitimate emergencies and not abused.

Acceptable Conditions

- Emergency functions require multisig approval
- Emergency actions are logged and transparent
- Automatic unpause after time limit is implemented

Trust Assumptions

- Trust that pause will only be used for legitimate emergencies
- Trust that admin will not abuse pause functionality
- Trust that pause will be lifted promptly after emergency

Other Governance & Trust

Timestamp Reliance BY DESIGN

• MINOR Reviewed — not scored

DESCRIPTION

Contract relies on block.timestamp for time-based logic — detected at SentaSampleVault.sol:93 in withdraw(). This introduces assumptions about validator behavior and network timing.

Acceptable Conditions

- Trust assumptions are explicitly documented
- Risk mitigation measures are implemented

Trust Assumptions

- Trust that validators will not manipulate block timestamps significantly
- Trust that network timing is sufficiently accurate for use case
- Trust that timestamp manipulation will not break contract logic

Privilege Review

Admin Controls

Upgradeable Status	Non-upgradeable
Admin Type	Owner-controlled
Timelock	Not detected
Multisig	Not detected

Privilege & Centralization Risks

Admin Controls

DESCRIPTION

Contract contains privileged functions controlled by owner/admin.

Threat Model

This is dangerous because if the admin private key is compromised (phishing, hardware wallet theft, or social engineering), an attacker gains immediate control over all contract funds and functionality. With no timelock or multisig protection, there is no delay or approval process to prevent malicious actions.

Conditional Impact

If the admin is compromised, then: 1. Attacker can immediately execute privileged functions. 2. Attacker can withdraw all funds in a single transaction. 3. Attacker can pause the contract or modify critical parameters. 4. Users have no recourse as privileged functions execute atomically.

Acceptable Conditions

This design may be acceptable only if:

- The contract holds minimal funds
- The admin key is stored in a hardware wallet with strong physical security
- The admin is a trusted entity with proven security practices
- The contract is experimental and not production-ready

Governance Risk

This is a design choice that concentrates power in a single admin role. For production contracts handling user funds, this level of centralization is unacceptable and should be mitigated with timelock or multisig. The absence of these protections creates a single point of failure.

Trust Assumptions

This design requires the following trust assumptions:

- The admin will not act maliciously
- The admin key will not be compromised
- The admin will follow responsible security practices
- The admin will not make mistakes that could lock or drain funds

Single Admin Control with Emergency Powers

RISK ▲ CRITICAL

Governance Risk

Classification: By Design

Full description, acceptable conditions, and trust assumptions for this characteristic are detailed in *Governance & Trust Observations*.

Global Pause Functionality

RISK ◆ MAJOR

Governance Risk

Classification: By Design

Full description, acceptable conditions, and trust assumptions for this characteristic are detailed in *Governance & Trust Observations*.

Security Posture & Risk Analysis

0 scored critical-severity code defects identified (1 critical-severity governance & trust risk reviewed, not scored). 0 scored major-severity code defects identified (1 major-severity governance & trust risk reviewed, not scored).

Trust Model:

- operational kill switches
- network and deployment assumptions
- privileged governance control

Security Posture Summary

This section provides system-level context on trust boundaries implied by privileged control surfaces and external dependencies observed in-scope.

Threat Model & Attack Surface

Timestamp Reliance

Contract relies on block.timestamp for time-based logic — detected at SentaSampleVault.sol:93 in withdraw(). This introduces assumptions about validator behavior and network timing.

Trust Assumptions & Inherited Risks

The following trust assumptions are implied by privileged control surfaces and external dependencies observed in-scope.

Trust assumptions apply across the following areas; each is detailed per-characteristic in *Governance & Trust Observations*:

- Governance Control
- Operational Killswitch
- Network Deployment

Operational & Environmental Exposure

Global Pause Functionality

Contract has global pause functionality that can halt all operations. This requires trust that pause will only be used for legitimate emergencies and not abused.

Systemic Risk Factors & Mitigations

The characteristics below are described in full under *Governance & Trust Observations*; this view focuses on their systemic mitigations.

Single Admin Control with Emergency Powers

- Implement timelock for critical operations
- Use multisig wallet for admin functions
- Add delay mechanisms for emergency functions
- Consider decentralized governance for critical decisions

Remediation Roadmap

The following prioritized action items outline the recommended remediation steps for all identified findings.

Priority	Finding	Action	Effort
1	GOV-01 **Single Admin Control with Emergency Powers** ▲ Critical	• Implement timelock for critical operations. • Use multisig wallet for admin functions. • Add delay mechanisms for emergency functions. • Consider decentralized governance for critical decisions.	High
2	GOV-02 **Global Pause Functionality** ◆ Major	• Implement timelock for pause (if possible). • Require multisig approval for pause. • Add automatic unpause after time limit. • Add transparency (events, notifications) for pause actions.	High
3	ST-001 **Address is not checked against the zero address** • Minor	• Validate address parameters are non-zero before assigning or using them.	Low
4	ST-002 **Exact equality check on a value others can influence** • Minor	• Avoid strict equality (==) on balances/timestamps/amounts an attacker can influence. • Use >= / <= threshold comparisons.	Low
5	ST-003 **Division before multiplication loses precision** • Minor	• Reorder arithmetic to multiply before dividing so integer truncation does not discard precision. • Otherwise scale by a fixed-point factor first.	Low
6	ST-004 **Logic depends on the block timestamp** • Minor	• Do not use block.timestamp for thresholds an attacker could game within validator tolerance. • Widen the window or anchor to block numbers.	Low
7	ST-005 **Local variable used before a value is set** • Minor	• Initialize all local variables before use.	Low
8	GOV-03 **Timestamp Reliance** • Minor	• Use block numbers for critical time-based logic. • Add tolerance ranges for timestamp comparisons. • Consider using oracle for precise time. • Document acceptable timestamp deviation.	High

Recommendations

No critical or major code defects were confirmed; remediation focuses on the 5 minor and 3 informational items below.

Across the 8 confirmed scored items, the assessment identified 8 distinct classes. The classes warranting code-level correction include division-before-multiplication precision loss; dangerous strict equality; missing zero-address validation; block-timestamp dependence; uninitialized local variable. Each should be addressed at its source and verified with targeted tests, following the specific guidance in the Detailed Findings for the corresponding entries. The remaining 3 lower-severity classes at the informational tier — such as inconsistent compiler versions across files, broad compiler version, unindexed address parameter in events — do not affect the grade but should be resolved to reduce ambiguity and strengthen maintainability, each with an explicit rationale where a control is intentionally retained.

Separately, 3 governance and operational-trust characteristics were reviewed and disclosed rather than scored as code defects; these do not require code remediation, but the team should document the associated trust assumptions and, where appropriate, apply operational controls — timelocks, multisig custody, and monitoring — proportionate to the authority they represent.

Looking beyond the immediate fixes, Senta Sample Vault would benefit from making security review a standing step in each release cycle and minimizing discretionary authority. The current result reflects a strong security posture that is worth preserving through disciplined change management. Establishing this discipline — with clear permission boundaries, deliberate change management, and transparent governance of privileged pathways — will help the system evolve safely as its ecosystem grows and reduce the likelihood of regressions in future integrations.

Conclusion

This audit assessed Senta Sample Vault under the SentaTrust Standard. The code received a SentaScore of A− (90/100), with 8 confirmed code defects. Governance risk is rated High: control over the contract is concentrated, and users rely on the operators acting as intended.

Next steps: fix the findings in the order of the Remediation Roadmap, starting with the highest severity. Where control is concentrated, document it openly for users and consider safeguards such as a timelock or a multisig.

The findings describe the code as submitted, at the time of the review. Later changes to the code, its dependencies or the way it is operated are not covered; a changed contract needs a new audit to be covered by a certificate.

This report supports informed decisions by the project team and everyone who relies on the contract. Decisions on fixes, deployment and operation remain with the project owners.

Technical Appendix

Audit & Scope Identity

Audit ID: SENTA-2026-990031
Audit Date: 05 Nov 2026
Network: Ethereum
Contract Address: Not applicable — source submission
Verification Link: Not applicable — source submission

Report Generation Metadata

Audit Package: Essential
Report Engine: SentaLabs Engine v3.3
Generated At: 06 Nov 2026, 09:40
Checksum (SHA256): acee61438b5de81376d42eb81ba424277f5d1e30feccd9dc57d0a9874007dd71

In-Scope Files

Path	Lines	Size
SentaSampleVault.sol	171	6.6 KB
SHA256 17e96cf23bf8e745c71a64e079874911be46a7a5472fd961065d117497dab222		

Submitted but Excluded Files

None

Findings Snapshot

The counts below are the **combined totals recorded in the findings record** — exactly what the integrity hash (SHA-256 above) attests — spanning **both** scored code defects and governance & trust observations. They are **not** the scored verdict; see the Findings Summary for the scored / governance partition that determines the SentaScore.

Total findings reviewed (combined across both axes): 11 — Critical: 1, Major: 1, Minor: 6, Informational: 3

Certification Sign-Off

This SentaShield audit was checked and approved by SentaTrust and issued under the SentaTrust Standard on 06 Nov 2026. The findings, severities and recommendations in this report describe the code as submitted.

SentaTrust

Certification Authority

Audit ID: SENTA-2026-990031

Date: 06 Nov 2026



SENTA-2026-990031

The SentaTrust Standard

SentaTrust issues every certification under a single, documented verification standard. Each report reflects an independent assessment of the reviewed code version against that standard, with all findings, severity ratings, and the SentaScore made available for independent verification. SentaTrust stands behind the integrity and consistency of this process for every certification it issues. A SentaTrust certification is a documented, verifiable record of that assessment — not a guarantee of the code's security.

Christian Krumrey
Founder & CEO

SentaTrust

Certified by SentaTrust

SentaTrust

Know for sure.

Questions about this report?

support@sentatrust.com · sentatrust.com

Scan to verify this certificate

acee61438b5de81376d42eb81ba424277f5d1e30fecc
d9dc57d0a9874007dd71

© 2026 Senta Trust Technologies LLC. All rights reserved.