HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Energy Web
Date: February 07, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Energy Web
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC1155; EIP-2535 Diamonds; Energy Resource Tracking
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://www.energyweb.org/
Changelog	30.12.2022 - Initial Review 07.02.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Energy Web (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://github.com/energywebfoundation/greenproof-sdk/tree/master/packages/greenproof-contracts
Commit	ce23f84f833eb5ce291458f92036db9c689e89c2
Whitepaper	https://www.energyweb.org/wp-content/uploads/2022/04/EW-Green-Proofs.pdf https://www.energyweb.org/case-studies-green-proofs/
Functional Requirements	GREENPROOFS-GreenProofsSmartContractFunctionalRequirements-201222-1109.pdf
Technical Requirements	https://github.com/energywebfoundation/greenproof-sdk/tree/master/packages/greenproof-contracts/docs/techspec.md
Contracts	File: ./packages/greenproof-contracts/contracts/facets/IssuerFacet.sol SHA3: c1eb94156bb601d99d697f28bce0ffd39f9b7a8bf83c891f4161959285ac3a04 File: ./packages/greenproof-contracts/contracts/facets/ProofManagerFacet.sol SHA3: 0518ed0c328feb349a630172b3a2d79fe72ad740e622413e406d5745af53bd0b File: ./packages/greenproof-contracts/contracts/facets/VotingFacet.sol SHA3: fc2e505d003a49d5faa74fa96fd0738c2a3138d753e7c0c2af2ebe1f2d102589 File: ./packages/greenproof-contracts/contracts/Greenproof.sol SHA3: 637a6d968e443174889f4f73a6b628a6c3548b45e0d4e48aabd16960e0ab19d4 File: ./packages/greenproof-contracts/contracts/interfaces/IClaimManager.sol SHA3: 117d7a5af63555884f0bc20a9c7a0a8afb6fb640aec928befdfad1e3c20f8e25 File: ./packages/greenproof-contracts/contracts/interfaces/IGreenProof.sol SHA3: 38eee9773b87fc2ffe5a1f33245d47751e55586142e495c9b7d9ad929addd74a File: ./packages/greenproof-contracts/contracts/interfaces/IProofManager.sol

	SHA3: 690d55afe3cdc90641cbe5c9e8b38c8916e8e3d3a473d1c8d14853c04ffb4f53 File: ./packages/greenproof-contracts/contracts/interfaces/IReward.sol SHA3: 70b846e555f59589fad71dbd7480007b010edc2b73251a627474cb709dfe0367 File: ./packages/greenproof-contracts/contracts/interfaces/IVoting.sol SHA3: 1d66f23dd5b52cee106507eaf9eb7843b64d1145c019f11d75170ba9176f6c98 File: ./packages/greenproof-contracts/contracts/libraries/LibClaimManager.sol SHA3: b937c12e1f4f6ba665dfbcb2a5bc924d0af13a1bc5b926b4c1a6c2e988999be3 File: ./packages/greenproof-contracts/contracts/libraries/LibIssuer.sol SHA3: d59d99add1a82b310c9ee76a6a7b5defbb070f33f58ac16ac72b0655702c692b File: ./packages/greenproof-contracts/contracts/libraries/LibProofManager.sol SHA3: d7092edf15157891321af59d4dac29723c5aa96bd11204ecf05551a017660a19 File: ./packages/greenproof-contracts/contracts/libraries/LibReward.sol SHA3: d4fedda548c1daaa444b6faea1a0e77b1027803da7483ff997f7084b22e2dd67 File: ./packages/greenproof-contracts/contracts/libraries/LibVoting.sol SHA3: 89f43eecd7d98703862f73cc3615888fbba1b94dbe4d774be7f0a2394eff35a5 File: ./packages/greenproof-contracts/contracts/upgradeInitializers/GreenproofInit.sol SHA3: ddc3e23ba973d8ee7c088f121924127cd8b9544ac6e4e8eb782bd6bc409c20fd

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Second review scope

Repository	https://github.com/energywebfoundation/greenproof-sdk/tree/master/packages/greenproof-contracts
Commit	4d9d110525e716da974f554ce8a29c003527d627
Contracts	File: ./packages/greenproof-contracts/contracts/facets/IssuerFacet.sol SHA3: 7983a8ff8afb58b226c7fcd59729b071cc98f07cdd54d39dc9e8aaaf0b67f5c8 File: ./packages/greenproof-contracts/contracts/facets/ProofManagerFacet.sol SHA3: baf88124af668b06477a32bd35daf6f1b1eb76d4afb5f9c30d1c17b1448e3637 File: ./packages/greenproof-contracts/contracts/facets/VotingFacet.sol SHA3: 096101661f1a4fc5ce0367035825457b4ccf4df8832766ef0c25696723a0d776 File: ./packages/greenproof-contracts/contracts/Greenproof.sol SHA3: b07ced81e28036093c1462abace75214d3b2aa01f25d4eba3e3551be302b3245 File: ./packages/greenproof-contracts/contracts/interfaces/IClaimManager.sol SHA3: 182ac1c34ab09d16faf55a0ed3c56d66ce415434faa42805d58662c62fd9c287 File: ./packages/greenproof-contracts/contracts/interfaces/IGreenProof.sol SHA3: 3f7d62f8008e1e9f3a32110513c33ea2e80eeab0b06e2206419fb18b67375d8c File: ./packages/greenproof-contracts/contracts/interfaces/IProofManager.sol SHA3: 8c75d45ed6e894477a1cc4ea5e80977c292ed9e90e104872cdd9c63078e6dd3f File: ./packages/greenproof-contracts/contracts/interfaces/IReward.sol SHA3: d11298c45e8a78f2f32629bd8bfea5cfc1666b6934682e2431b6959691627df3 File: ./packages/greenproof-contracts/contracts/interfaces/IVoting.sol SHA3: 0e6a1cf81b0d28be541f701e9018d96678fd44cb076135f7fe4c1dfc64ba9e47 File: ./packages/greenproof-contracts/contracts/libraries/LibClaimManager.sol SHA3: 106586485dd689894542a55bdbeeffb840a5d4164e2535fee7c639cf6c8cef2f File: ./packages/greenproof-contracts/contracts/libraries/LibIssuer.sol SHA3: c1f2bb454e5f26498a6132830480604a5e4b9ffa72d3bafcee4561e1939ec128

	File: ./packages/greenproof-contracts/contracts/libraries/LibProofManager.sol SHA3: fa77886382cd54e5d45368aec1a60a9f4609949e516457d49b82d356dbf42a3b File: ./packages/greenproof-contracts/contracts/libraries/LibReward.sol SHA3: 0b19d838137d90b44baa9dce5c9f9808e9602697bf63b01254b0d4bbffd76008 File: ./packages/greenproof-contracts/contracts/libraries/LibVoting.sol SHA3: 6b5f25f705f9416cb0c40431065162d10cf10c3abcb91f0e13d75a28703e65bc File: ./packages/greenproof-contracts/contracts/upgradeInitializers/GreenproofInit.sol SHA3: 7c803c8a8ff02058504556f4862346b2a195c7da2207f274a21adc3e630cc277

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is detailed.
- NatSpec is consistent.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code follows official language style guides and best practices.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions by several users are tested thoroughly.

Security score

As a result of the audit, the code contains **3** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**

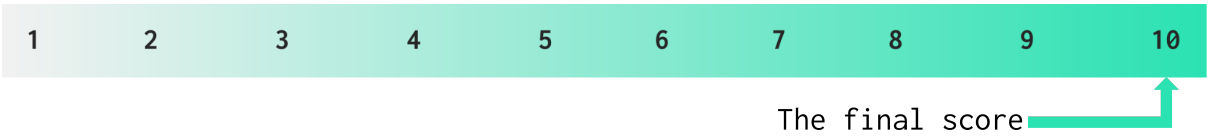


Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
30 December 2022	16	8	1	0
07 February 2023	3	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant

Token Supply Manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

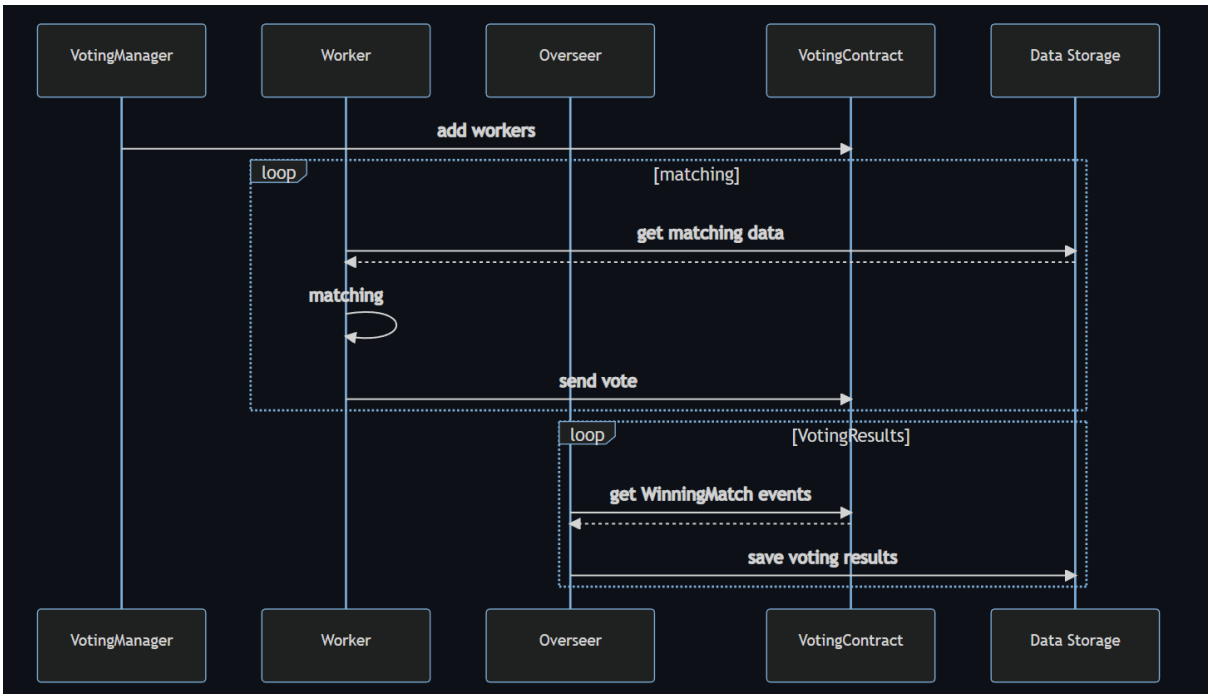
Greenproof contracts are used to issue certificates assuring certain properties of electricity generators. These certificates are used to determine how much electricity parties have left in their disposal according to the amount they have and have not used.

Certificated data is verified by voting among trusted parties. Only those parties who have acquired the necessary verifiable credentials are allowed to participate in the voting. To avoid security risks, no actual data is stored on-chain.

The project aims to achieve a trustable approach on the usage of clean energy, by keeping track of the energy usage of parties that are participating in the system.

Identifier of generation is a hash of generation data and it is called `inputHash`, generation data is represented by its Merkle tree and is called `matchResult`.

The system flow, including off-chain part of the project, is given as the following diagram:



Information about the contracts in the scope:

- **Greenproof.sol:** High level contract which plays two roles:
  - to proxy calls to business logic components.
  - to initialize and configure logic components.

- **GreenproofInit.sol:** Initializes the necessary supported interfaces on ERC165 for contracts IClaimManager, IGreenProof, IProofManager, IVoting. Used on deployment of Greenproof.sol.
- **IClaimManager.sol:** Interface for Claim Manager Contract and Claim Revocation Registry, which are used inside LibClaimManager.sol contract *hasRole()* function.
- **IGreenProof.sol:** Interface representation of IssuerFacet.sol contract.
- **IProofManager.sol:** Interface representation of ProofManagerFacet.sol contract.
- **IRewards.sol:** Interface representation of VotingFacet.sol contract. Responsible for rewards related functions.
- **IVoting.sol:** Interface representation of VotingFacet.sol contract. Responsible for voting related functions.
- **VotingFacet.sol:** Exposes API for driving and configuration of the voting. Internal methods are separated into LibVoting library. Since voting is identified by input hash, it has been aliased as votingID.
- **LibVoting.sol:** Contains voting functions used by facets.
- **IssuerFacet.sol:** Contract with main purpose of minting tokens certifying energy attributes. The basis for certification is voting for (votingID, matchResult), which is completed with consensus. Second purpose of the contract is token transferring.
- **LibIssuer.sol:** Contains certificate functions shared by facets.
- **ProofManagerFacet.sol:** Contract claiming certified energy generation.
- **LibProofManager.sol:** Contains claim verification functions shared by facets.
- **LibClaimManager.sol:** Contains functions, which verify authorization in facets.
- **LibReward.sol:** Library managing reward payment.

Privileged roles

- **Owner:**
  The only account allowed to upgrade contracts, add or remove participants of voting, cancel expired votings and enable or disable reward of voters, who have reached consensus.
- **Worker:**
  Account who is able to determine generator data by generation identifier. Since this process requires off-chain data workers are off-chain nodes. Only nodes which are able to present certain verifiable credentials are allowed to vote.
- **Issuer:**
  Is authorized to mint energy certificates and make their data public.
- **Revoker:**
  Any user with the revokerRole (defined during contract deployment) is allowed to revoke a certificate.

- **Claimer:**
  Any user with the claimerRole (defined during contract deployment) is allowed to retire either partially or totally a certificate on behalf of the certificate owner.

Risks

- Iterating over a dynamic array populated with a custom number of variables can lead to Gas limit denial of service if the size of array goes out of control.
- The system is fully centralized and upgradeable. The audit only covers contract versions within the scope and does not cover the generation and storage model of private keys. We recommend that all privileged role accounts be at least ⅗ multi-sig.
- The @solidstate module was audited by Hacken, but since it has been some time since that audit, it should be considered a possible risk and evaluated accordingly.
- Greenproof depends on external Claim Manager And Claim Revocation Registry as a source of the roles in the system which are not part of the scope.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Data Consistency

In the LibVoting.sol library in the *_getVoters()* function, the size of the *_voters* array will be allocated based on the voting session's vote count.

In the case where the worker is removed from the whitelisted list after they voted, but before the consensus, the size of the allocated array will be bigger than the remaining valid workers.

This leads to an array with empty elements being returned, and thus part of the rewards being transferred to *address(0)*.

Path:
./contracts/libraries/LibVoting.sol : _getVoters()

Recommendation: Update the *_getVoters()* function to only return an array containing valid voters.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

■■ Medium

M01. Denial of Service (DoS)

In the VotingFacet.sol contract, in the *cancelExpiredVotings()* function, a nested *for* loop that depends on variable array lengths is used.

These arrays are not limited in length and will continue to grow exponentially throughout the contract's lifecycle.

This may result in Denial of Service in the corresponding functions if the array gets too large to the point that Gas needed to execute and iterate through the arrays get larger than the maximum Gas the EVM allows.

Path:
./contracts/facets/VotingFacet.sol : cancelExpiredVotings()

Recommendation: The function can take index parameters that allows the transaction of the for loop to be divided into multiple transactions with lower Gas costs.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

M02. Denial of Service (DoS)

In the LibReward.sol contracts, the *payReward()* function uses a while loop that depends on the length of an array. This array is not limited in length and is determined by the number of workers added to the *rewardQueue* array.

This may result in Denial of Service in the corresponding functions if the array gets too large to the point that Gas needed to execute and iterate through the arrays get larger than the maximum Gas the EVM allows.

Path:
./contracts/libraries/LibReward.sol : payReward()

Recommendation: Consider introducing a limitation mechanism for the while loop.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

M03. Contradiction - Documentation Mismatch

In the comment in the VotingFacet.sol contract's *removeWorker()* function, it is stated that only the contract owner can execute the function. However, there is no such limitation in the code, which is in line with the provided documentation.

This can result in unexpected contract behavior and lowers the code readability.

Path:
./contracts/facets/VotingFacet.sol : removeWorker()

Recommendation: Fix the mismatch between comment and the documentation.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

M04. Requirements Violation

It is stated in the documentation that if a consensus is not reached before the time limit during voting, the vote is expired. Further vote casting should restart the session, and all previous votes should be canceled.

There is no such functionality within the code.

After the expiration, the session is ended with a *Completed* status, but without the reward payment. The *NoConsensusReached()* and *VotingSessionExpired()* events are emitted during this process.

Path:
./contracts/facets/VotingFacet.sol : vote()

Recommendation: Consider updating the code to follow the documentation, or updating the requirements within the documentation to match the code.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

M05. Contradiction - Missing Functionality

Greenproof contracts use external contracts, deployed and managed by the Energy Web Foundation, for managing roles and revoking roles within the Greenproof system.

Claim Manager and Claim Revocation Registry contract addresses are provided as parameters during the deployment of the Greenprof contract.

The Owner can update the Claim Manager address using the *updateClaimManager()* function, but there is no functionality to update the Claim Revocation Registry address.

Path:
./contracts/Greenproof.sol

Recommendation: Consider adding functionality for updating the Claim Revocation Registry address and update the documentation with information about Claim Revocation Registry functionality.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

M06. Inconsistent Data

It is considered best to keep any data as accurate as possible until losses are minimized.

The function *_getAmountHash()* is dividing its parameter volume by 1 Ether and transforming the result into a string.

In cases where volume is not divided directly into integers, the resulting string will be incorrect due to the loss of precision.

As the string is used to generate the volumeHash, which is used as a leaf in Merkle Tree proof verification, the loss of precision may lead to a Denial of Service in the *requestProofIssuance()* function of the IssuerFacet.sol contract.

Path:
./contracts/libraries/LibIssuer.sol : _getAmountHash()

Recommendation: Consider reversing the requirements and providing the amount for *requestProofIssuance()* as an integer without 18 decimals, and performing multiplication by 1 Ether in the mint function to mitigate the possibility of a DoS vulnerability.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

M07. Best Practice Violation

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The built-in *transfer()* and *send()* functions process a hard-coded amount of Gas. If the receiver is a contract with a *receive* or *fallback* function, the transfer may fail due to an "out of Gas" exception.

This could lead to denial of service situations for specific accounts.

Paths:
./contracts/libraries/LibReward.sol : payReward()
./contracts/libraries/LibVoting.sol : _rewardWinners()

Recommendation: Follow common best practices and replace *transfer()* and *send()* functions with *call()*, or provide a special mechanism for interacting with a smart contract.

Status: Mitigated
(with Customer notice, the transfer addresses are all EOA, so it is believed that there is no risk of the receiver being a contract.)

M08. Contradiction - Missing Validation

Inside the *requestProofIssuance()* function, the minting of the ERC1155 certificates is done without checking if the target is a contract that is an ERC1155Receiver implementer.

This contradicts how the *safeTransferFrom()* and *safeBatchTransferFrom()* are implemented.

This can lead to a situation where certificates are minted to the smart contract that do not support the ERC1155 standard and the locking of certificates.

Path:
./contracts/facets/IssuerFacet.sol : requestProofIssuance()

Recommendation: Consider using _safeMint() for secure certificates issuance.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

Low

L01. Floating Pragma

Locking the pragma helps to ensure that contracts are not accidentally deployed using an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:
./contracts/Greenproof.sol

./contracts/upgradeInitializers/GreenproofInit.sol
./contracts/libraries/LibClaimManager.sol
./contracts/libraries/LibIssuer.sol
./contracts/libraries/LibProofManager.sol
./contracts/libraries/LibReward.sol
./contracts/libraries/LibVoting.sol
./contracts/interfaces/IClaimManager.sol
./contracts/interfaces/IGreenProof.sol
./contracts/interfaces/IProofManager.sol
./contracts/interfaces/IReward.sol
./contracts/interfaces/IVoting.sol
./contracts/facets/IssuerFacet.sol
./contracts/facets/ProofManagerFacet.sol
./contracts/facets/VotingFacet.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L02. State Variable Default Visibility

The visibility of bytes32 constant variables is not explicitly set in library contracts.

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Paths:
./contracts/libraries/LibClaimManager.sol
./contracts/libraries/LibIssuer.sol
./contracts/libraries/LibReward.sol
./contracts/libraries/LibVoting.sol

Recommendation: Variables can be specified as being public, internal, or private. Explicitly define visibility for all state variables.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L03. Inefficient Gas Model

In the LibReward.sol contracts *payReward()* function, the variable *rs.rewardAmount* is used inside the while loop instead of the previously declared *rewardAmount* variable, which has the same value.

Reading from storage costs more Gas than reading from memory, resulting in inefficient Gas usage.

Path:
./contracts/libraries/LibReward.sol : payReward()

Recommendation: Use *rewardAmount* variable instead of *rs.rewardAmount*.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L04. Style Guide Violation

The project should follow the official code style guidelines.
Inside each contract, library, or interface, use the following order:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

Within a grouping, place the view and pure functions at the end.

Solidity style guidance defines a naming convention that should be followed. Some state variables are not in the mixed case.

Some contracts are not formatted correctly.

Paths:
./contracts/facets/ProofManagerFacet.sol
./contracts/facets/VotingFacet.sol
./contracts/libraries/LibClaimManager.sol
./contracts/libraries/LibVoting.sol

Recommendation: The official Solidity style guidelines should be followed.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L05. Unfinished NatSpec

It is recommended that the code should be kept clean and properly documented with NatSpec. There are multiple functions, structs, and public storage variables that are missing proper NatSpec documentation.

Paths:
./contracts/Greenproof.sol
./contracts/upgradeInitializers/GreenproofInit.sol
./contracts/libraries/LibClaimManager.sol
./contracts/libraries/LibIssuer.sol
./contracts/libraries/LibProofManager.sol
./contracts/libraries/LibReward.sol
./contracts/libraries/LibVoting.sol

./contracts/interfaces/IClaimManager.sol
./contracts/interfaces/IGreenProof.sol
./contracts/interfaces/IProofManager.sol
./contracts/interfaces/IReward.sol
./contracts/interfaces/IVoting.sol
./contracts/facets/IssuerFacet.sol
./contracts/facets/ProofManagerFacet.sol
./contracts/facets/VotingFacet.sol

Recommendation: NatSpec documentation best practices should be followed. For reference:
https://docs.soliditylang.org/en/v0.8.17/natspec-format.html#documentation-example
https://dev.to/perelynsama/natspec-the-right-way-to-comment-ethereum-smart-contracts-1b0c

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L06. Unindexed Events

Having indexed event parameters makes it easier to search for these events using indexed event parameters as filters.

Paths:
./contracts/interfaces/IVoting.sol : VotingSessionExpired()
./contracts/libraries/LibVoting.sol : WinningMatch(), NoConsensusReached(), MatchRegistered(), ConsensusReached()

Recommendation: The “indexed” keyword should be used for the event parameters.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L07. Redundant Imports

The use of unnecessary imports will increase the Gas consumption of the code. Thus, they should be removed from the code.

Redundant imports decrease code readability.

Paths:
./contracts/libraries/LibIssuer.sol : IVoting, ERC1155BaseInternal, ERC1155EnumerableInternal
./contracts/libraries/LibProofManager.sol : ERC1155BaseStorage
./contracts/interfaces/IGreenProof.sol : LibIssuer
./contracts/facets/IssuerFacet.sol : IVoting
./contracts/facets/VotingFacet.sol : LibIssuer

Recommendation: Consider removing redundant code.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L08. Redundant Use of Override Specifier

Starting from the 0.8.8 version, a function that overrides only a single interface function does not require the *override* specifier.

The use of redundant *override* specifiers decreases the code readability.

Paths:
./contracts/facets/IssuerFacet.sol : requestProofIssuance(), discloseData(), getCertificateOwners()
./contracts/facets/ProofManagerFacet.sol: claimProof(), revokeProof(), getProof(), getProofIdByDataHash(), getProofsOf(), claimedBalanceOf(), verifyProof()
./contracts/facets/VotingFacet.sol : addWorker(), removeWorker(), cancelExpiredVotings(), getWorkers(), getWinners(), numberOfVotings(), replenishRewardPool()

Recommendation: Consider removing redundant code.

Status: Reported
(IssuerFacet.sol contracts *discloseData()* and VotingFactes.sol contracts *cancelExprideVotings()* override specifiers remains)

L09. Code Consistency

It is best practice to write code uniformly.

There is no consistency in how reverts are handled or in the messages for those reverts in the Greenproof contracts.

In one case, custom *errors* are used; in another *requires*.

Paths:
./contracts/facets/IssuerFacet.sol
./contracts/facets/ProofManagerFacet.sol
./contracts/facets/VotingFacet.sol

Recommendation: Be consistent with the approach to reverting and the messages sent when reverting.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L10. Redundant Validation

There are redundant *require()* statements in the *safeTransferFrom()* and *safeBatchTransferFrom()* functions.

There is no need to revert with "*invalid zero token ID*" or "*tokenId greater than issuer.latestCertificateId*" as transfers will always revert with "*insufficient balance*" under the same conditions.

Using additional checks increases the Gas consumption of the call and decreases the code clarity.

Path:
./contracts/facets/IssuerFacet.sol: safeTransferFrom(), safeBatchTransferFrom()

Recommendation: Consider removing redundant *require()* statements.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L11. Code Consistency

It is best practice to write code uniformly.

There is no consistency in how data is accessed in functions.

In one case, an internal library function is used; in another, data is accessed directly from the storage variable.

"*session.workerToVoted[msg.sender] == true*" vs "*LibVoting._hasAlreadyVoted(worker, session)*" "*LibVoting._isClosed(session)*" vs "*session.status == LibVoting.Status.NotStarted*"

Path:
./contracts/facets/VotingFacet.sol

Recommendation: Be consistent with the approach to reverting and the messages sent when reverting.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L12. Inefficient Gas Model

In the *getWinningMatches()* function the storage variable *voting.sessionIDs.length* is read twice.

Reading from storage costs more Gas than reading from memory, resulting in inefficient Gas usage.

Path:
./contracts/facets/VotingFacet.sol : getWinningMatches()

Recommendation: Reorder the function; declare *numberOfVotingSessionIds* before the *winningSessionsIDs*. Use *numberOfVotingSessionIds* for *winningSessionsIDs* array length.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

L13. Duplicated Event Declarations

There are multiple cases of duplicate *event* declarations.

Code duplication decreases code readability.

Paths:
./contracts/interfaces/IGreenProof.sol : ProofMinted()

./contracts/interfaces/IReward.sol : RewardsActivated(), RewardsDeactivated()
./contracts/interfaces/IVoting.sol : WinningMatch(), NoConsensusReached(), MatchRegistered(), ConsensusReached()
./contracts/libraries/LibVoting.sol : VotingSessionExpired()

Recommendation: Remove duplicated code.

Status: Reported
(The ProofMinted() event is declared both in IGreenProof and LibIssuer contracts. The other duplicate code is removed.)

L14. Unused Code

There are multiple cases of unused code declarations.

Unused code declarations decrease code readability.

Paths:
./contracts/libraries/LibClaimManager.sol : NotInitializedClaimManager()
./contracts/libraries/LibIssuer.sol : DEFAULT_VCREDENTIAL_VALUE
./contracts/libraries/LibReward.sol : matchVotingAddress

Recommendation: Remove duplicated code.

Status: Reported
(*NotInitializedClaimManager()* remains, others are removed.)

L15. Redundant Storage Variable

The *voteToCertificates* mapping from the IssuerStorage is redundant.

It contains the same data as the *dataToCertificateID* mapping.

It is never accessed from the code, nor is any of its data exposed by an external view function.

Storing any data inside *voteToCertificates* mapping only increases the gas consumption of the *_registerProof* function.

Path:
./contracts/libraries/LibIssuer.sol : voteToCertificates

Recommendation: Remove redundant *voteToCertificates* storage variable and all code connected to it.

Status: Mitigated
(with Customer notice, the voteToCertificates is a reserved storage slot for future development.)

L16. Code Consistency

It is best practice to write code uniformly.

There is no consistency in prefixes for internal and private function names in the library contracts.

The underscore _ prefix is used without any obvious reason.

Paths:
./contracts/libraries/LibClaimManager.sol
./contracts/libraries/LibIssuer.sol
./contracts/libraries/LibProofManager.sol
./contracts/libraries/LibReward.sol
./contracts/libraries/LibVoting.sol

Recommendation: Consider following best practices and only use an underscore prefix for private library functions, and name the internal functions without the underscore.

Status: Fixed
(Revised commit: 4d9d110525e716da974f554ce8a29c003527d627)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.