Dust Labs

DeBridge

SECURITY ASSESSMENT

25.03.2023

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Dust Labs. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (14.03.2023)	Layout
0.4 (15.03.2023)	Automated Security Testing Manual Security Testing
0.5 (16.03.2023)	Verify Claims and Test Deployment
0.9 (17.03.2023)	Summary and Recommendation
1.0 (17.03.2023)	Final document
1.1 (25.03.2023)	Re-check

2. Project Overview

DUST protocol is a decentralized protocol and SPL (Solana Program Library) token created on the Solana blockchain with a starting supply of zero, and a maximum supply of 33,300,000. DUST has an emission schedule with multiple halvenings and mining rewards that are earned via staking NFTs. Countless projects have independently adopted DUST within their own ecosystems making it the most used SPL token on the Solana blockchain.

DeGods and Y00ts are two popular nonfungible token (NFT) digital art collections on the Solana blockchain. Both of these projects have attracted attention from the NFT community, offering unique digital artwork and novel features for collectors.

Website:
https://degods.com
https://www.y00ts.com

Twitter:
https://twitter.com/DeGodsNFT
https://twitter.com/y00tsNFT

Discord:
https://discord.gg/dedao
https://discord.gg/y00ts

Instagram:
https://www.instagram.com/thedegods
https://instagram.com/they00ts

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the smart contract functioning in a number of scenarios, or creates a risk that the smart contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using the smart contract, or provides the opportunity to use the smart contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the smart contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the smart contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of experts, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. 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.
   ii. Symbolic execution, which is analysing the smart contract to determine what inputs causes each part of the smart contract to execute.
3. Best practices review, which is a review of the smart contract to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure the smart contract.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Program Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 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

File	Fingerprint (MD5)
./solana/programs/dust_bridging/src/instructions/burn_and_send.rs	b9dcff9ae0422420765226d06430f078
./solana/programs/dust_bridging/src/instructions/initialize.rs	7638a5c511f07005ebf66b328c8f0064
./solana/programs/dust_bridging/src/instructions/mod.rs	5af503510a902e2f5c0c63ac305b8edf
./solana/programs/dust_bridging/src/instructions/admin.rs	0c09ce3a91afe994181077a9832ffb55
./solana/programs/dust_bridging/src/error.rs	8775580de6cc684c5d67c2ea32e04ad5
./solana/programs/dust_bridging/src/lib.rs	02c494646cae269f4084192c7543e7e4
./solana/programs/dust_bridging/src/instance.rs	260c2335fa8d6495052d928ae31c0bb2
./solana/programs/dust_bridging/src/anchor_metadata.rs	30ecb4025871b57ff0978a4eebe11c05
./evm/src/nft/DustWormholeERC721Upgradeable.sol	1941631f87c34de96595edc28c7b5a5e
./evm/src/nft/DefaultOperatorFiltererUpgradeable.sol	8a2ac64cdf656c987599f61164355c24
./evm/src/nft/OperatorFiltererUpgradeable.sol	1a89064002849f73e08e11f7909b2899
./evm/src/nft/IOperatorFilterRegistry.sol	b94aae7a13345e7a6335c36449130f53
./evm/src/nft/lib/Constants.sol	876f88aa97e2dd0ad7d1fdccea191a00

5.2 Source Unites in Scope

Total : 13 files, 664 codes, 214 comments, 167 blanks

Files

filename	language	code	comment	blank	total
dust-bridging/evm/src/nft/DefaultOperatorFiltererUpgradeable.sol	Solidity	8	7	3	18
dust-bridging/evm/src/nft/DustWormholeERC721Upgradeable.sol	Solidity	180	32	42	254
dust-bridging/evm/src/nft/IOperatorFilterRegistry.sol	Solidity	27	88	25	140
dust-bridging/evm/src/nft/OperatorFiltererUpgradeable.sol	Solidity	43	30	8	81
dust-bridging/evm/src/nft/lib/Constants.sol	Solidity	3	1	2	6
dust-bridging/solana/programs/dust_bridging/src/anchor_metadata.rs	Rust	42	1	10	53
dust-bridging/solana/programs/dust_bridging/src/error.rs	Rust	6	0	1	7
dust-bridging/solana/programs/dust_bridging/src/instance.rs	Rust	32	2	5	39
dust-bridging/solana/programs/dust_bridging/src/instructions/admin.rs	Rust	54	0	15	69
dust-bridging/solana/programs/dust_bridging/src/instructions/burn_and_send.rs	Rust	167	41	31	239
dust-bridging/solana/programs/dust_bridging/src/instructions/initialize.rs	Rust	47	9	11	67

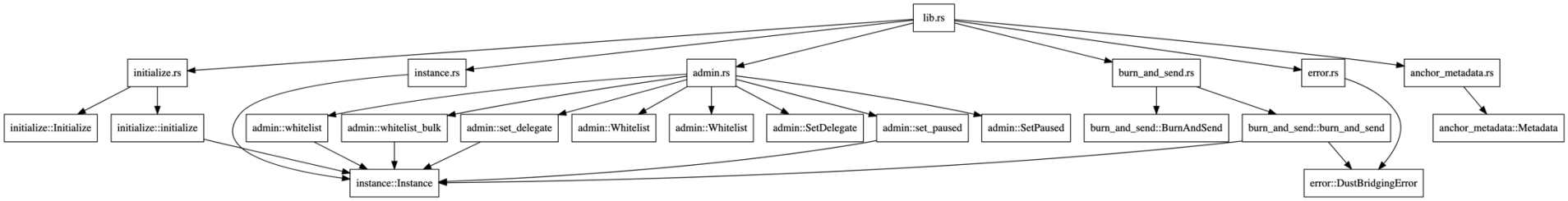
filename	language	code	comment	blank	total
dust-bridging/solana/programs/dust_bridging/src/instructions/mod.rs	Rust	6	0	2	8
dust-bridging/solana/programs/dust_bridging/src/lib.rs	Rust	49	3	12	64

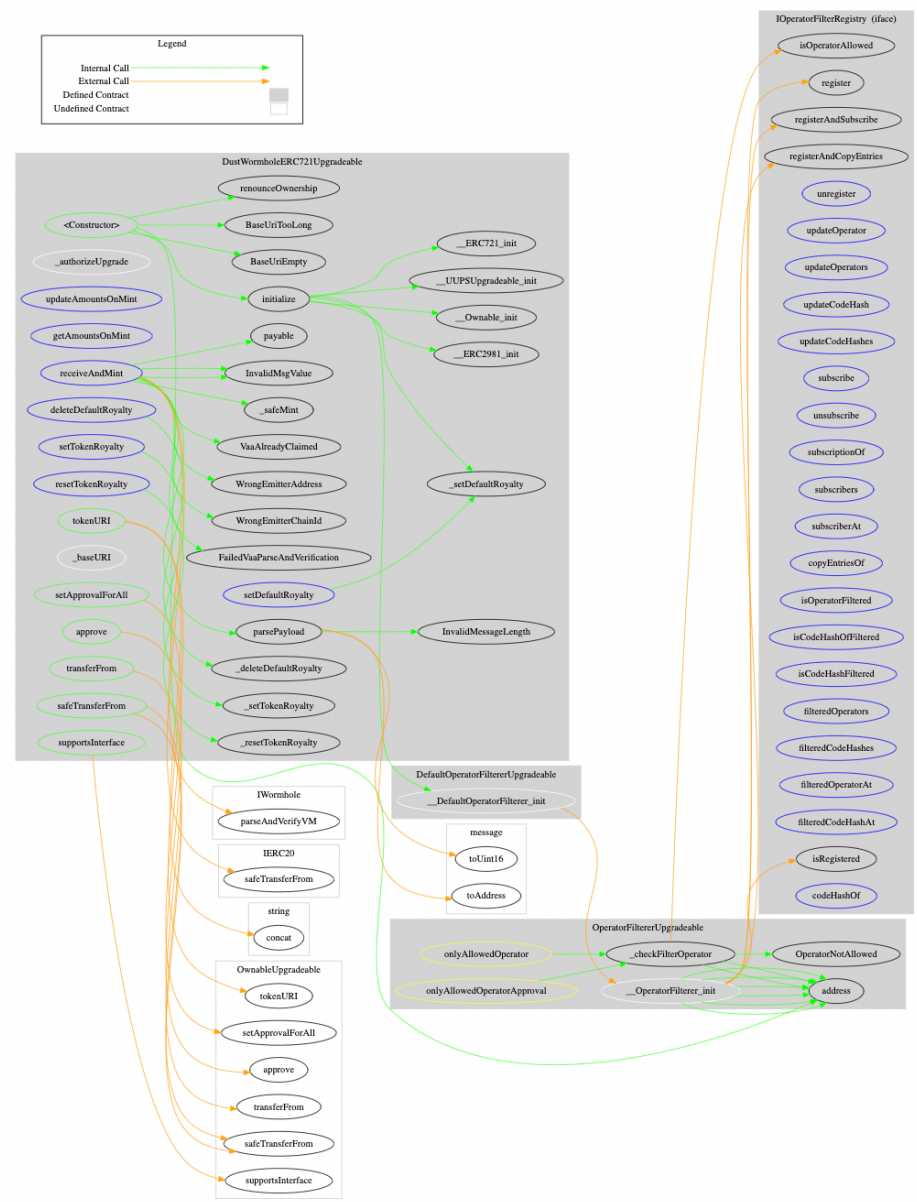
5.3 Used Code from other Frameworks/Dependencies

Dependency / Import Path	Source
@certusone/wormhole-sdk	https://www.npmjs.com/package/@certusone/wormhole-sdk/v/0.9.11
@metaplex-foundation/mpl-token-metadata	https://www.npmjs.com/package/@metaplex-foundation/mpl-token-metadata/v/2.9.1
@project-serum/anchor	https://www.npmjs.com/package/@project-serum/anchor/v/0.26.0
@solana/spl-token	https://www.npmjs.com/package/@solana/spl-token/v/0.3.7
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC20/utils/SafeERC20.sol
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/master/contracts/proxy/utils/UUPSUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/master/contracts/token/ERC721/ERC721Upgradeable.sol

Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/token/common/ERC2981Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/master/contracts/token/common/ERC2981Upgradeable.sol
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/master/contracts/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/master/contracts/proxy/utils/Initializable.sol
wormhole-solidity/BytesLib.sol	https://github.com/wormhole-foundation/wormhole/blob/main/ethereum/contracts/libraries/external/BytesLib.sol
wormhole-solidity/IWormhole.sol	https://github.com/wormhole-foundation/wormhole/blob/main/ethereum/contracts/interfaces/IWormhole.sol

5.4 Graph


lib.rs
initialize.rs
instance.rs
admin.rs
burn_and_send.rs
error.rs
anchor_metadata.rs
initialize::Initialize
initialize::initialize
admin::whitelist
admin::whitelist_bulk
admin::set_delegate
admin::Whitelist
admin::Whitelist
admin::SetDelegate
admin::set_paused
admin::SetPaused
burn_and_send::BurnAndSend
burn_and_send::burn_and_send
anchor_metadata::Metadata
instance::Instance
error::DustBridgingError


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
DustWormholeERC721Upgradeable
renounceOwnership
<Constructor>
BaseUriTooLong
_authorizeUpgrade
BaseUriEmpty
updateAmountsOnMint
initialize
getAmountsOnMint
payable
receiveAndMint
InvalidMsgValue
deleteDefaultRoyalty
_safeMint
setTokenRoyalty
VaaAlreadyClaimed
resetTokenRoyalty
WrongEmitterAddress
tokenURI
WrongEmitterChainId
_baseURI
FailedVaaParseAndVerification
setApprovalForAll
setDefaultRoyalty
approve
parsePayload
InvalidMessageLength
transferFrom
_deleteDefaultRoyalty
safeTransferFrom
_setTokenRoyalty
supportsInterface
_resetTokenRoyalty
__ERC721_init
__UUPSUpgradeable_init
__Ownable_init
__ERC2981_init
_setDefaultRoyalty
IWormhole
parseAndVerifyVM
IERC20
safeTransferFrom
string
concat
OwnableUpgradeable
tokenURI
setApprovalForAll
approve
transferFrom
safeTransferFrom
supportsInterface
DefaultOperatorFiltererUpgradeable
__DefaultOperatorFilterer_init
message
toUint16
toAddress
OperatorFiltererUpgradeable
onlyAllowedOperator
_checkFilterOperator
OperatorNotAllowed
onlyAllowedOperatorApproval
__OperatorFilterer_init
address
IOperatorFilterRegistry (iface)
isOperatorAllowed
register
registerAndSubscribe
registerAndCopyEntries
unregister
updateOperator
updateOperators
updateCodeHash
updateCodeHashes
subscribe
unsubscribe
subscriptionOf
subscribers
subscriberAt
copyEntriesOf
isOperatorFiltered
isCodeHashOfFiltered
isCodeHashFiltered
filteredOperators
filteredCodeHashes
filteredOperatorAt
filteredCodeHashAt
isRegistered
codeHashOf

6. Scope of Work

The Dust Labs developer team provided us with the files that needs to be tested. The scope of the audit is the xchain wormhole bridge, called DustBridging.

The team put forward the following assumptions regarding the security, usage of the contracts:

Purpose

This project implements a Solana program called DustBridging which facilitates the bridging of Dust's DeGods and y00ts NFT collections from Solana to Ethereum and Polygon respectively using Wormhole for the cross-chain step. It also provides a TypeScript SDK to interact with the on-chain program.

Design Summary

DustBridging is a program on the Solana blockchain written with the Anchor framework.

Burn and Send

Its most important instruction is called burn_and_send which burns a provided NFT and emits a Wormhole message, thus initiating the bridging process.
In more detail, when invoked, it will:

1. Ensure that all its prerequisites are fulfilled, namely that

- the NFT belongs to the collection of the given instance of DustBridging
- the instance isn't paused
- the NFT is whitelisted (if whitelisting is enabled)

2. Additionally it relies on Metaplex's new Burn instruction to ensure that:

- the NFT is a verified item of the collection
- the transaction was signed by the owner of the NFT or an authorized delegate and is hence authorized to burn the NFT
- the NFT is the master edition and not some other edition
- that a coherent set of Metaplex accounts was provided

3. Burn the NFT.
4. Emit a Wormhole message which serves as proof for the burning of the NFT and which can be submitted on the target EVM chain to mint its equivalent there. The Wormhole message contains a tokenId (2 bytes) (taken from the URI of the metadata field (see example NFTs at the bottom of this readme)) and an EVM address (20 bytes) of the designated recipient's wallet.

Admin Instructions

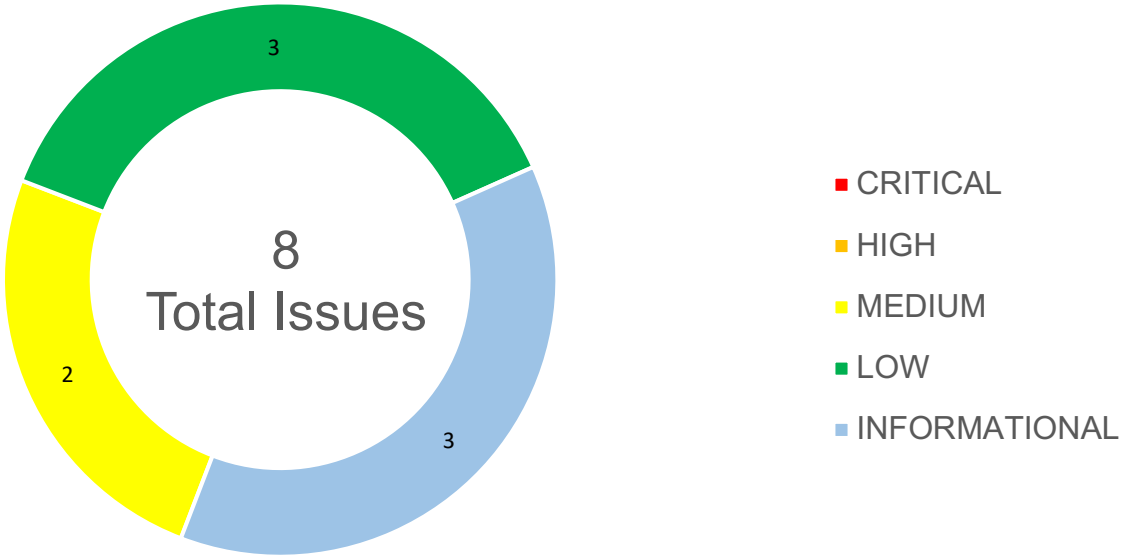
The program can be instantiated multiple times but only once per Collection NFT and only by the UpdateAuthority of that collection (who can then be thought of as the admin of that program instance) by using the initialize instruction.
DustBridging supports:

- an optional whitelist -- Passing a collection size argument of 0 to the initialize instruction disables the whitelist, otherwise it must be set to the size of the collection (there is no way to undo an initialization that used the wrong collection size argument!).
- whitelisting (whitelist and whitelist_bulk) -- whitelist sets the corresponding bit of an NFT with the given token id to true and is hence more natural, while whitelist_bulk allows writing directly to the underlying bit array for a more efficient approach (primarily inteded for setting up the initial state of the whitelist).
- delegating (set_delegate) -- Allows delegating admin functionality to a separate account (known as the delegate).
- pausing (set_paused) -- So burnAndSend instructions will fail even if all other prerequisites are met.

Note: The audit does not include the wormhole logic, protocol, or platform itself. That part of the bridge process is considered as out of scope.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Potential Loss Of Ownership	MEDIUM	FIXED
6.2.2	Handle Edge Case In whitelist_bytes Function	MEDIUM	FIXED
6.2.3	Missing State Variable Visibility	LOW	FIXED
6.2.4	Improve Registry Contract Detection Using ERC165	LOW	FIXED
6.2.5	Out-of-Bounds Access Risk	LOW	FIXED
6.2.6	Floating Pragma	INFORMATIONAL	ACKNOWLEDGED
6.2.7	if-else Statement	INFORMATIONAL	ACKNOWLEDGED
6.2.8	Descriptive Error Message	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting's experts found **0 Critical issues** in the code of the program.

HIGH ISSUES
During the audit, Chainsulting's experts found **0 High issues** in the code of the program.

MEDIUM ISSUES
During the audit, Chainsulting's experts found **3 Medium issues** in the code of the program.

6.2.1 Potential Loss Of Ownership
Severity: MEDIUM
Status: FIXED
Code: CWE-282
File(s) affected: DustWormholeERC721Upgradeable.sol
Update: Fixed with Ownable2StepUpgradeable

Attack / Description	The owner of a NFT token contract can be changed by calling transferOwnership function of OpenZeppeling Ownable implementation. This function directly sets the owner to the given address, if the address is not the zero address. Making such a critical change in a single step is error-prone and can lead to irrevocable mistakes.
Code	Line 9 (DustWormholeERC721Upgradeable.sol) import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

Result/Recommendation	It is recommended to use the two step pattern by setting the new owner as potential owner in the first step and approving the ownership transfer by calling an approve function by the new owner. OpenZeppelin provides such as Ownable2Step implementation. This prevents transferring ownership to an invalid address.

6.2.2 Handle Edge Case In whitelist_bytes Function

Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: initialize.rs
Update: Fixed with proposed implementation

Attack / Description	The whitelist_bytes function might produce a wrong size for the whitelist vector when the collection_size is not divisible by 8, potentially causing issues in the whitelist vector's functionality.

Code	Line 14 - 15 (initialize.rs)
	`const fn whitelist_bytes(collection_size: u16) -> usize {`
	`  (collection_size/8 + if (collection_size % 8) > 0 {1} else {0}) as usize`
	`}`

Result/Recommendation	Update the function to handle the edge case properly by adding 7 to collection_size before dividing it by 8. This will ensure that the whitelist vector has the correct size. const fn whitelist_bytes(collection_size: u16) -> usize { ((collection_size + 7) / 8) as usize } The reason for updating the whitelist_bytes function is to handle the case where the collection_size is not divisible by 8. In the current implementation, if the collection_size is not divisible by 8, the resulting whitelist vector size will be smaller than it should be. This is because integer division in Rust will truncate the decimal part, effectively rounding down the result. For example, if the collection_size is 10, the current implementation will calculate the whitelist vector size as 10 / 8 = 1.25, which will be truncated to 1.

	However, a vector of size 1 can only store 8 bits of information, which is insufficient to represent the whitelist state of a collection with a size of 10. By adding 7 to the collection_size before dividing it by 8, you ensure that the resulting whitelist vector size is always rounded up, providing enough space to store the whitelist state for the entire collection. In the example above, the updated function will calculate the size as (10 + 7) / 8 = 2.125, which will be truncated to 2, creating a vector with enough space to store the whitelist state for all 10 items in the collection. In summary, updating the whitelist_bytes function as recommended will properly handle the edge case where collection_size is not divisible by 8, ensuring that the whitelist vector has enough space to store the state for the entire collection.

LOW ISSUES

During the audit, Chainsulting's experts found **3 Low issues** in the code of the program.

6.2.3 Missing State Variable Visibility

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: DustWormholeERC721Upgradeable.sol

Attack / Description	In the current implementation several state variables are missing an explicit visibility modifier. By default all state variables have public visibility and auto generate view functions.

Code	Line 37 – 52 (DustWormholeERC721Upgradeable.sol)
	// Core layer Wormhole contract.
	IWormhole immutable _wormhole;
	// ERC20 DUST token contract.
	IERC20 immutable _dustToken;
	// Contract address that can mint NFTs. The mint VAA should have this as the emitter addr
	bytes32 immutable _minterAddress;

	// Common URI for all NFTs handled by this contract.
	bytes32 immutable _baseUri;
	uint8 immutable _baseUriLength;
	
	// Amount of DUST to transfer to the minter on upon relayed mint.
	uint256 _dustAmountOnMint;
	// Amount of gas token (ETH, MATIC, etc.) to transfer to the minter on upon relayed mint.
	uint256 _gasTokenAmountOnMint;
	// Dictionary of VAA hash => flag that keeps track of claimed VAAs
	mapping(bytes32 => bool) _claimedVaas;

Result/Recommendation	It is recommended to check, if the default behaviour of state variables (public getter function) is desired and explicitly mark the state variables visibility.

6.2.4 Improve Registry Contract Detection Using ERC165

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: OperatorFiltererUpgradeable.sol

Attack / Description	Enhance the detection of the registry contract by leveraging the ERC165 standard for interface detection Currently, the OperatorFiltererUpgradeable contract checks the presence of the registry contract by verifying the code length at a specific address. This method is not the most reliable or recommended way to determine if a contract is present or implements a specific interface. The ERC165 standard provides a more reliable approach to detect the presence of a contract and whether it supports a particular interface.
Code	Line 19 – 20 (OperatorFiltererUpgradeable.sol) `IOperatorFilterRegistry constant OPERATOR_FILTER_REGISTRY = IOperatorFilterRegistry(0x000000000000AAeB6D7670E522A718067333cd4E);`

Result/Recommendation	It is recommended by extending the IOperatorFilterRegistry interface to implement IERC165 and updating the OperatorFiltererUpgradeable contract to utilize the ERC165 standard, which can enhance the robustness of the contract detection process.
	
	Update IOperatorFilterRegistry to extend IERC165:
	import "@openzeppelin/contracts/utils/introspection/IERC165.sol";
	
	interface IOperatorFilterRegistry is IERC165 {
	  // The rest of your functions
	}
	
	Implement supportsInterface in the OperatorFilterRegistry contract:
	
	function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
	  return interfaceId == type(IOperatorFilterRegistry).interfaceId || super.supportsInterface(interfaceId);
	}
	
	Replace the registry contract detection code in OperatorFiltererUpgradeable.sol:
	
	// Replace Line 20 in OperatorFiltererUpgradeable.sol
	IERC165 constant OPERATOR_FILTER_REGISTRY_INTERFACE =
	  IERC165(0x000000000000AAeB6D7670E522A718067333cd4E);
	
	function _checkFilterOperator(address operator) internal view virtual {
	  // Check if the contract at the given address supports the IOperatorFilterRegistry interface using ERC165
	  if (
	  address(OPERATOR_FILTER_REGISTRY_INTERFACE).code.length > 0 &&
	
	OPERATOR_FILTER_REGISTRY_INTERFACE.supportsInterface(type(IOperatorFilterRegistry).interfaceId)

	) {
	IOperatorFilterRegistry registry =
	IOperatorFilterRegistry(address(OPERATOR_FILTER_REGISTRY_INTERFACE));
	if (!registry.isOperatorAllowed(address(this), operator)) {
	revert OperatorNotAllowed(operator);
	}
	}
	}

6.2.5 Out-of-Bounds Access Risk

Severity: LOW
Status: FIXED
Code: CWE-787
File(s) affected: instance.rs

Attack / Description	The is_whitelisted function in instance.rs might access an out-of-bounds index in the whitelist vector if token_id is greater than or equal to collection_size. This could lead to unexpected behavior or errors if an invalid token ID is provided that is outside the range of the collection size.

Code	Line 36 – 38 (instance.rs)
	
	`pub fn is_whitelisted(&self, token_id: u16) -> bool {`
	`self.whitelist[token_id as usize / 8] & (1u8 << (token_id % 8)) > 0`
	`}`

Result/Recommendation	Add a check to ensure token_id is within the valid range of the collection size before accessing the whitelist vector. You can accomplish this by validating token_id against the collection size before proceeding with the function.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **3 Informational issues** in the code of the program.

6.2.6 Floating Pragma
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: ALL

Attack / Description	The current pragma Solidity directive is floating. It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
Code	pragma solidity ^0.8.13;
Result/Recommendation	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. i.e. Pragma solidity 0.8.13 See SWC-103: https://swcregistry.io/docs/SWC-103

6.2.7 if-else Statement
Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: OperatorFiltererUpgradeable.sol

Attack / Description	Improve code readability by splitting if-else statement into separate if statements with clear comments In the OperatorFiltererUpgradeable.sol contract, the nested if-else statement in lines 29-37 might be confusing and difficult to read.

Code	Lines 29 - 37 (OperatorFiltererUpgradeable.sol)
	{
	// If an inheriting token contract is deployed to a network without the registry deployed, the modifier
	// will not revert, but the contract will need to be registered with the registry once it is deployed in
	// order for the modifier to filter addresses.
	if (address(OPERATOR_FILTER_REGISTRY).code.length > 0) {
	if (!OPERATOR_FILTER_REGISTRY.isRegistered(address(this))) {
	if (subscribe) {
	OPERATOR_FILTER_REGISTRY.registerAndSubscribe(address(this), subscriptionOrRegistrantToCopy);
	} else {
	if (subscriptionOrRegistrantToCopy != address(0)) {
	OPERATOR_FILTER_REGISTRY.registerAndCopyEntries(address(this), subscriptionOrRegistrantToCopy);
	} else {
	OPERATOR_FILTER_REGISTRY.register(address(this));
	}
	}
	}
	}
	}

Result/Recommendation	Split the nested if-else statement into separate if statements and provide clear comments for each condition. By applying these changes, the code becomes more readable, allowing for easier understanding and maintenance.

6.2.8 Descriptive Error Message
Severity: INFORMATIONAL
Status: ACKNOWLEDGED

Code: NA
File(s) affected: OperatorFiltererUpgradeable.sol

Attack / Description	Improve consistency and debugging by using a require statement with a descriptive error message In the OperatorFiltererUpgradeable.sol contract, the onlyAllowedOperator modifier uses a custom error instead of a require statement with a descriptive error message. To make the code more consistent with the rest of the contract and improve debugging, consider using a require statement with a descriptive error message.

Code	Lines 48 - 64 (OperatorFiltererUpgradeable.sol)
	
	`modifier onlyAllowedOperator(address from) virtual {`
	`// Allow spending tokens from addresses with balance`
	`// Note that this still allows listings and marketplaces with escrow to transfer tokens if transferred`
	`// from an EOA.`
	`if (from != msg.sender) {`
	`_checkFilterOperator(msg.sender);`
	`}`
	`_;`
	`}`

Result/Recommendation	Replace the custom error with a require statement that includes a descriptive error message:
	
	// line 57 in OperatorFiltererUpgradeable.sol (modified)
	if (from != msg.sender) {
	require(OPERATOR_FILTER_REGISTRY.isOperatorAllowed(address(this), msg.sender), "Operator not allowed");
	}
	
	By applying this change, the code becomes more consistent with the rest of the contract and provides better error messages for debugging purposes.

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the solana rust program and evm solidity smart contracts.

The primary objective of the audit was to evaluate the security and functionality of the smart contracts. Throughout the audit process, our team identified no critical issues, no high-severity issues, 1 medium-severity issue, 3 low-severity issues, and 3 informational issues. These results were obtained through a combination of manual and automated security testing.

The code quality observed throughout the audit is generally high, with well-structured and organized code. The use of modular design, clear function naming, and adherence to Solidity and Rust best practices contribute to the overall readability and maintainability of the project. Additionally, the implementation of the proxy pattern and upgradeable contracts demonstrates the team's foresight in planning for future updates and enhancements.

However, there are some areas where the code quality could be further improved. Adding more comments and documentation for public functions and structures will make it easier for others to understand the code and will aid in maintaining and updating the code in the future. Addressing the identified vulnerabilities and recommendations will also contribute to a higher overall code quality.

Update (25.03.2023): The DustLabs Team fixed all important issues and improved the overall code quality

8. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive Web3 solutions. Their services include Web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of many top DeFi projects.

Chainsulting currently secures $100 billion in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: https://chainsulting.de