SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

DubIr

Audit

Security Assessment

20. August, 2022

For

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	19. June 2022	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	20. August 2022	• Reaudit

Network
Binance Smart Chain (BEP20)

Website
https://github.com/dublr/dublr

Twitter
https://twitter.com/DublrToken

Description

Dublr is a smart contract token that implements several token standards (ERC20, ERC777, ERC1363, ERC4524, EIP2612). It has its own built-in distributed exchange (so it is both a token and a DEX). Supply is generated on-demand by minting, with a mint price that grows exponentially.

Project Engagement

During the 28th of June 2022, **Dublr Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

- Github
  - https://github.com/dublr/dublr
  - Commit: b84ea8edd8c1ab6a3f1abbc34467fa2ce4b21526

v1.1

- Github
  - https://github.com/dublr/dublr
  - Commit: c72a4d0c21e8c05e52124f919f129946f91355fa

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 contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the 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 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

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 expert pentesters and smart contract developers, documenting any issues as there were discovered.

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 SolidProof 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 SolidProof 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 a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts 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 your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

```
./OmniTokenInternal.sol
./interfaces/IERC20.sol
./interfaces/IERC20Optional.sol
./interfaces/IERC20Burn.sol
./interfaces/IERC20SafeApproval.sol
./interfaces/IERC20IncreaseDecreaseAllowance.sol
./interfaces/IERC20TimeLimitedTokenAllowances.sol
./interfaces/IERC777.sol
./interfaces/IERC1363.sol
./interfaces/IERC4524.sol
./interfaces/IEIP2612.sol
```

```
./DublrInternal.sol
./interfaces/IDublrDEX.sol
```

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

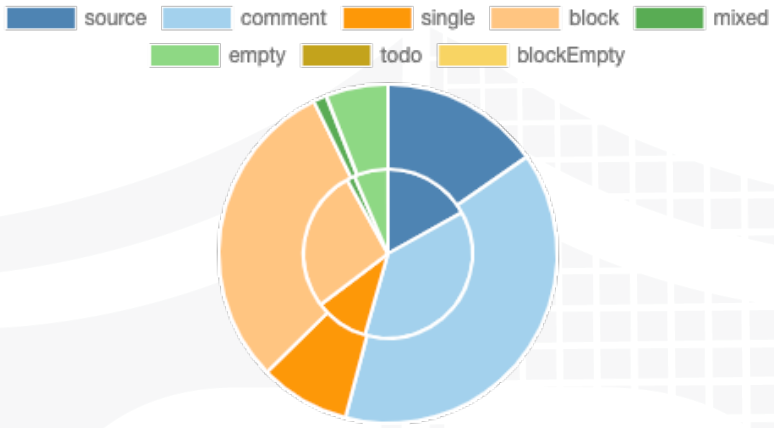
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

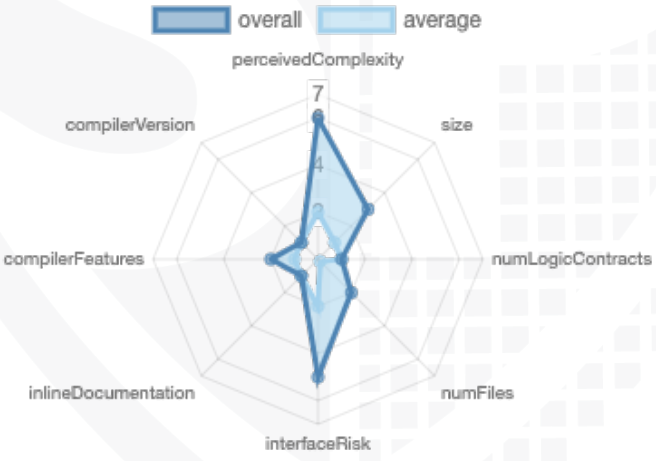
File Name	SHA-1 Hash
contracts/interfaces/IERC20SafeApproval.sol	2bf1ffd209e753ee18387e7138bac80cee4fd52a
contracts/interfaces/IERC165.sol	c1a81e4178a593af653547eb3c8413be7b05256b
contracts/interfaces/IERC777.sol	ed86b896b0a87db25aca22b4901f2a7086e754ac
contracts/interfaces/IERC4524.sol	31d61018d8c3347087eed8c03da161b5a1966aab
contracts/interfaces/IDublrDEX.sol	b206a69ec987bf300d3ea331615310a72477519d
contracts/interfaces/IERC20TimeLimitedTokenAllowances.sol	21efb6305790ca4cf1b668d34e9cd082b91b4753
contracts/interfaces/IERC1363.sol	0d8c7d46cb5cf0ba445cc6091e184579ce957d40
contracts/interfaces/IERC20Optional.sol	da5fe2de67be283abc0a9cb553dbf4aba8217dd9
contracts/interfaces/IEIP2612.sol	8cbd253b69deace35d873f58ed69f80a0612df12
contracts/interfaces/IERC20Burn.sol	b55027d4dc24a9bee10773fb1638155b2025e962
contracts/interfaces/IERC20IncreaseDecreaseAllowance.sol	0b0037c912e4e64341c879394c5819340c228e1b
contracts/interfaces/IERC20.sol	f86885444d5a76a2a1e59179c50de37812472506
contracts/DublrInternal.sol	0765f28b07a7ff230b9dfbf427a9bb1e07e8d670
contracts/OmniTokenInternal.sol	62fcbd1aaaa3f18ffbe4e6e49944369c438b8327
contracts/OmniToken.sol	9fb2819d8a2419ba726abd4105f89268ec184f97
contracts/Dublr.sol	a73da24ba2605e8dd09f80b6449273385e8c7700

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	0	12	2

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Version	Public	Payable
1.0	102	4

Version	External	Internal	Private	Pure	View
1.0	92	120	7	7	35

State Variables

Version	Total	Public
1.0	50	10

Capabilities

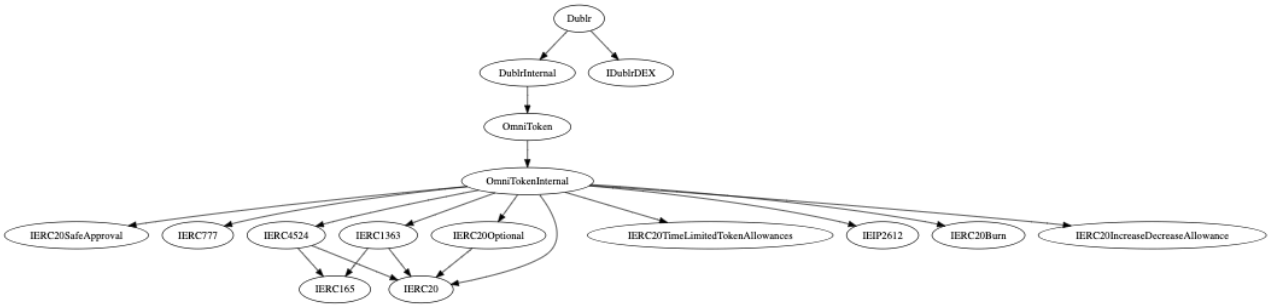
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.8.15`		`yes`	`yes` (6 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/Create/Create2

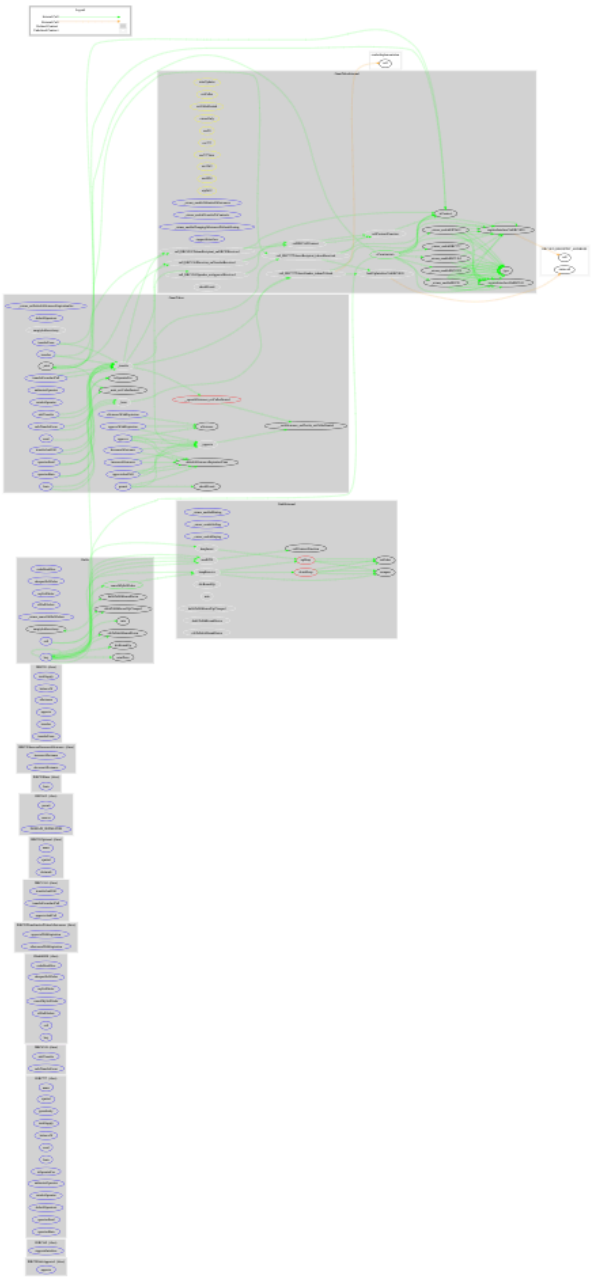
1.0				yes	yes	

Inheritance Graph

v1.0



CallGraph
v1.0



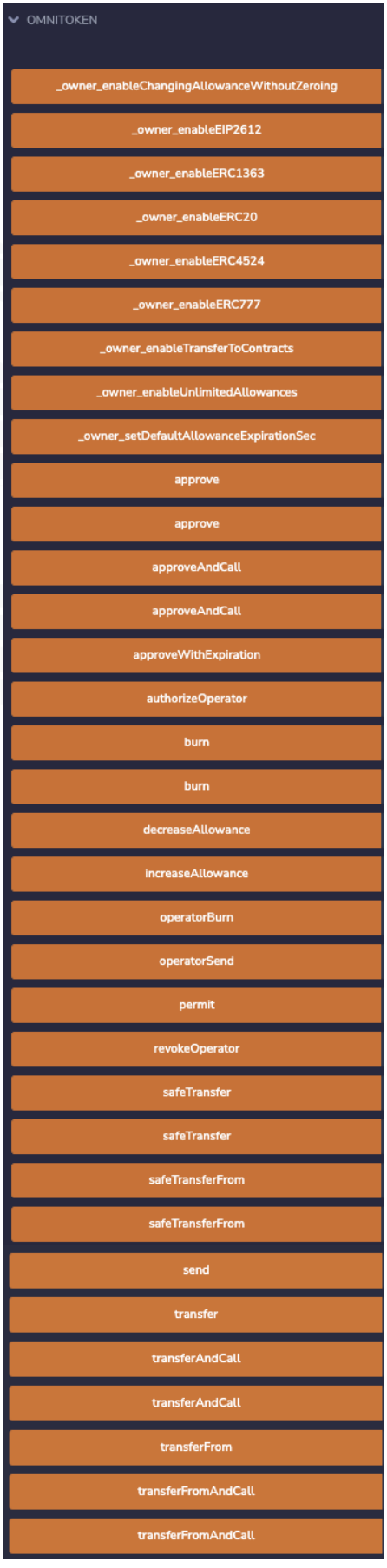
Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Overall checkup (Smart Contract Security)

Write functions of contract v1.0



Overall checkup (Smart Contract Security)

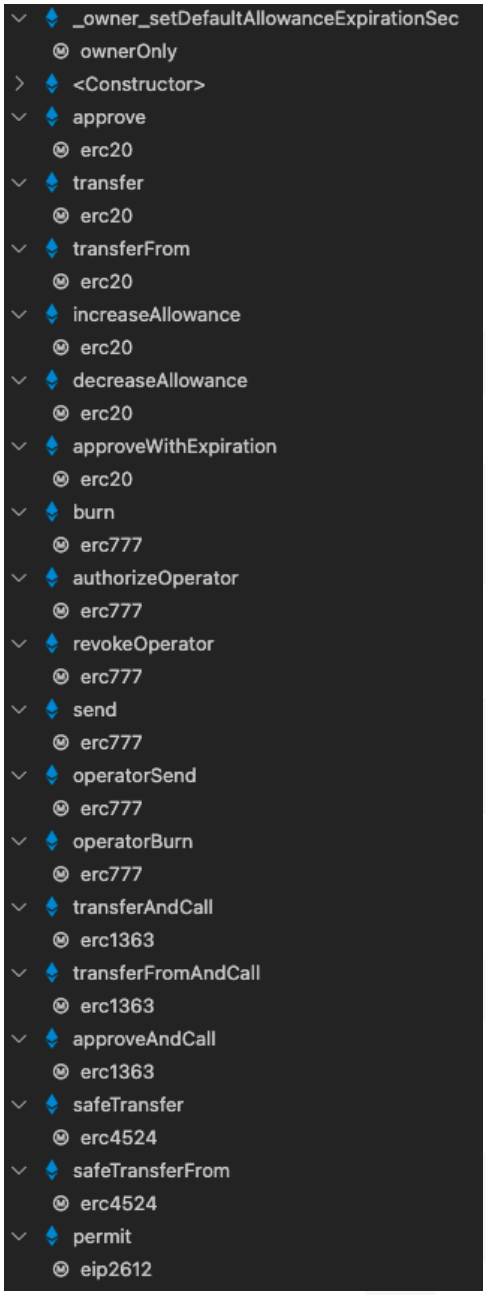
Tested	Verified
✓	✓

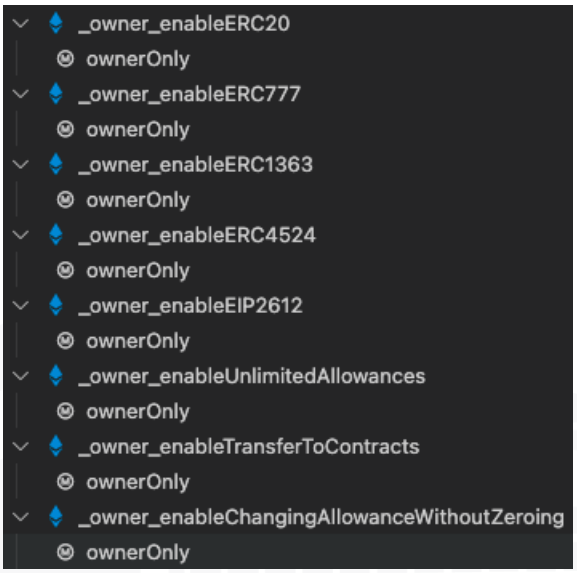
Legend

Attribute	Symbol
Verfified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	-

Modifiers and public functions

v1.0





Comments

- *Deployer can enable/disable following state variables*
  - mintingEnabled
  - sellingEnabled
  - buyingEnabled

- *Existing Modifiers*
  - stateUpdater
  - extCaller
  - extCallerDenied
  - ownerOnly

    - erc20
    - erc777
    - erc777View
    - erc1363
    - erc4524
    - eip2612

- Addresses are able to transfer/burn for another address as an operator
- Owner can enable
    - ERC20
    - ERC777
    - ERC1363
    - EIP2612
    - Unlimited allowances
    - Transfer contracts
    - Changing allowance without zeroing enabled

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope
v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/interfaces/IERC20SafeApproval.sol	———	1	75	73	5	64	3	———
🔍	contracts/interfaces/IERC165.sol	———	1	20	18	3	13	3	🔆
🔍	contracts/interfaces/IERC777.sol	———	1	225	79	9	183	27	———
🔍	contracts/interfaces/IERC4524.sol	———	1	110	34	5	93	13	———
🔍	contracts/interfaces/IDublrDEX.sol	———	1	278	111	16	230	23	💰
🔍	contracts/interfaces/IERC20TimeLimitedTokenAllowances.sol	———	1	73	53	5	59	5	———
🔍	contracts/interfaces/IERC1363.sol	———	1	162	42	5	142	17	———
🔍	contracts/interfaces/IERC20Optional.sol	———	1	24	14	4	12	9	———
🔍	contracts/interfaces/IEIP2612.sol	———	1	48	40	3	37	7	🔆
🔍	contracts/interfaces/IERC20Burn.sol	———	1	23	21	3	16	3	———
🔍	contracts/interfaces/IERC20IncreaseDecreaseAllowance.sol	———	1	68	38	4	57	5	———
🔍	contracts/interfaces/IERC20.sol	———	1	136	38	5	115	13	———
💊	contracts/DublrInternal.sol	1	———	345	342	124	184	68	Σ
💊	contracts/OmniTokenInternal.sol	1	———	795	766	311	367	275	🧱 🔨 Σ
📝	contracts/OmniToken.sol	1	———	1349	1255	312	929	267	🧱 Σ
📝	contracts/Dublr.sol	1	———	728	724	225	439	152	💰 Σ
📝🔍💊	**Totals**	**4**	**12**	**4459**	**3648**	**1039**	**2940**	**890**	💻 💰 🧱 🔨 🔆 Σ

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
nSLOC	normalized source lines of code (only source-code lines; no comments, no blank lines)
Comment Lines	lines containing single or block comments
Complexity Score	a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Audit Results

AUDIT PASSED

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	Main	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities

Informational issues

No informational issues

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

20 August 2022:

- Read whole report and modifiers section for more information

Test Protocol

OmniToken

- ✓ ERC20: Constant functions
- ✓ ERC20: totalSupply
- ✓ ERC20: balanceOf
- ✓ ERC20: Transfer adds amount to destination account and subtracts from sender account
- ✓ ERC20: Can transfer full balance
- ✓ ERC20: Cannot transfer more than balance
- ✓ ERC20: Test disabling API
- ✓ ERC20: Cannot transfer from empty account
- ✓ ERC20: Transfer emits event
- ✓ ERC20: Set allowance and send to own wallet
- ✓ ERC20: Set allowance and send to other wallet
- ✓ ERC20: Cannot transferFrom without allowance
- ✓ ERC20: Unlimited allowance
- ✓ ERC20 extension: increaseAllowance / decreaseAllowance
- ✓ ERC20 extension: set allowance with expected current value
- ✓ ERC20 extension: allowanceWithExpiration
- ✓ ERC20 extension: burn
- ✓ ERC777: send function should revert for non-ERC777 contract recipient
- ✓ ERC777: send function should succeed for EOA recipient
- ✓ ERC777: send function should call ERC777 recipient, with non-ERC777 sender
- ✓ ERC777: send function should call ERC777 sender interface if present
- ✓ ERC777: test reentrancy protection
- ✓ ERC777: burn
- ✓ ERC777: authorizeOperator / revokeOperator
- ✓ ERC777: operatorSend
- ✓ ERC777: operatorBurn
- ✓ ERC1363: transferAndCall
- ✓ ERC1363: transferFromAndCall
- ✓ ERC1363: transferFromAndCall to EOA should fail
- ✓ ERC1363: approveFromAndCall
- ✓ ERC4524: safeTransfer
- ✓ ERC4524: safeTransferFrom
- ✓ ERC4524: safeTransferFrom to EOA should succeed
- ✓ EIP2612: permit

Dublr

- ✓ Minting can be disabled, but by owner only
- ✓ Mint price
- ✓ Minting without any sell orders
- ✓ Only one sell order at once
- ✓ Sell orders are sorted
- ✓ Sell orders can be bought
- ✓ Larger sell orders
- ✓ Roll over from one sell order to the next when an order is exhausted
- ✓ Buying transitions from buying sell orders to minting at mint price
- ✓ Mint price over 1.0 ETH per DUBLR
- ✓ Unpayable seller
- ✓ Unpayable buyer

✓ Heap stress test

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED