World of Dypians - Audit

Security Assessment

CertiK Assessed on Nov 29th, 2024

CertiK Assessed on Nov 29th, 2024

World of Dypians - Audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
GameFi	Binance Smart Chain (BSC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 11/29/2024	N/A

CODEBASE	COMMITS
WoD-Contracts	828217ac151d57fd3cb73350a8c3a4c72b6b91ea
View All in Codebase Page	View All in Codebase Page

Vulnerability Summary

5	2	1	1	1	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	1 Resolved, 1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	1 Partially Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
0	Informational		Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | WORLD OF DYPIANS - AUDIT

CODEBASE | WORLD OF DYPIANS - AUDIT

Repository

WoD-Contracts

Commit

828217ac151d57fd3cb73350a8c3a4c72b6b91ea

AUDIT SCOPE | WORLD OF DYPIANS - AUDIT

2 files audited ● 1 file with Acknowledged findings ● 1 file with Partially Resolved findings

ID	Repo	File	SHA256 Checksum
● WOC	worldofdypians/WoD-Contracts	WorldOfDypians.sol	7a239349adc11175365c9bae70d1b8213d85cbf7a81756ef0c226cf588339bbd
● VCW	worldofdypians/WoD-Contracts	Vesting.sol	b498be8a03106f32856c01e4ec9d46835b29b8fabd5afa16df22df5f1fc6a117

APPROACH & METHODS | WORLD OF DYPIANS - AUDIT

This report has been prepared for World of Dypians to discover issues and vulnerabilities in the source code of the World of Dypians - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | WORLD OF DYPIANS - AUDIT

5	0	2	1	2	0
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for World of Dypians - Audit. Through this audit, we have uncovered 5 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

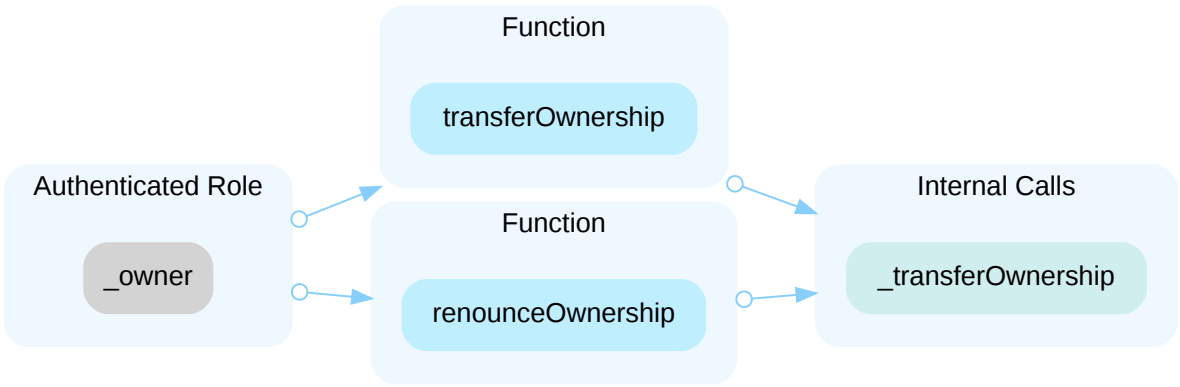
ID	Title	Category	Severity	Status
GLOBAL-01	**Centralization Related Risks**	**Centralization**	**Major**	**Resolved**
WOC-02	**Initial Token Distribution**	**Centralization**	**Major**	**Mitigated**
WOC-01	No Public Mint Function	Logical Issue	Medium	Resolved
VCW-03	Missing Checks On Function `addVestingWallets`	Volatile Code	Minor	Partially Resolved
WOC-04	Pull-Over-Push Pattern In `transferOwnership()` Function	Logical Issue	Minor	Acknowledged

GLOBAL-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location
Centralization	**Major**	

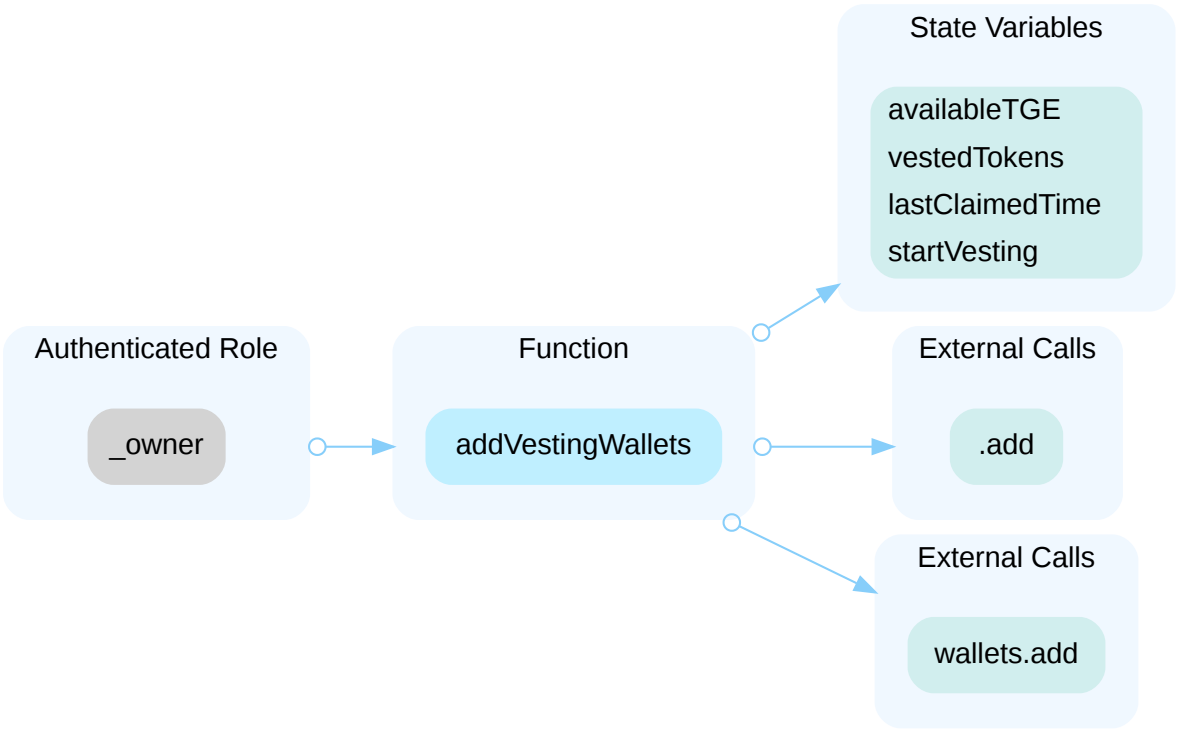
Description

In the contract `Ownable` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` acc
renounce, transfer the ownership.



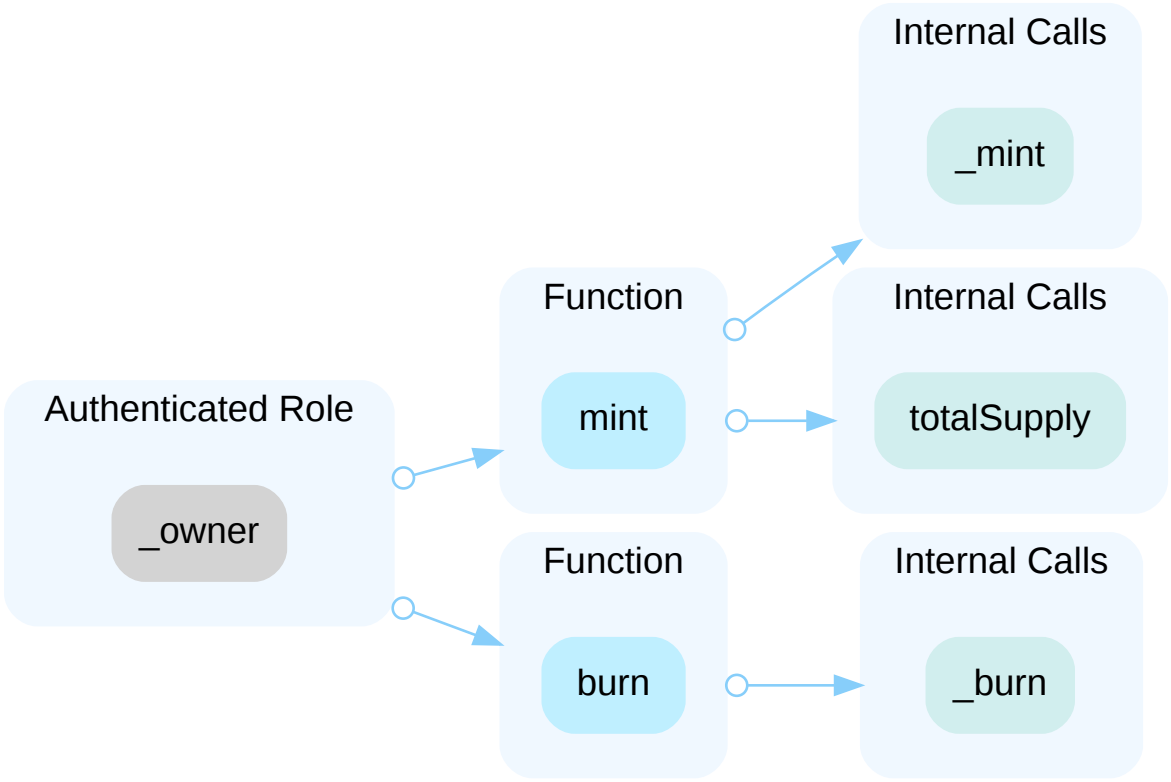
In the contract `TokenVestingLock` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_`

- `addVestingWallets` : adding wallets into the Vesting Contract



In the contract `WorldOfDypians` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_ow`

- `mint` : mint the tokens
- `burn` : burn the tokens



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failu

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[World of Dypians Team, 11/28/2024]: The owner role of the WoD Token contract deployed at address https://bscscan.com/address/0xb994 zero address.

Renounce transaction: https://bscscan.com/tx/0xbb44f16c11b4e7f8d5bfa37d129d9af2396f3d6fbe42c5b7b1d10189dbd2392d.

We have renounced the Ownership of all Vesting Contracts BSC:

CATEGORY	ADDRESS	
Seed	0xD62FC589701C1FC54675124C42fe1D7fF4e0204C	0x5b6e763b046490f38ccdf5
Private	0x0A3C5eE8F6F7b552E436f922e4F3a28E24343f7b	0x7cbc5852347ad4fa1e7fc3
Private-IDO	0x9f149D2d422a12Ba34bee11473863625B9793B66	0xca49a50efd98481298e82
KOL	0xaD07ef12F836409FF0d7206860Fd0174F7Bda342	0xda83c40c21ab051d79865
Team	0x218b135F64669cb102F0810A49E64E437C85c9F2	0x508057007cc441348a436
Advisors	0x255b1C2e3f2FF180d45f1e055224d97b23079513	0x8ed69a948c9c6b70ff9592
Community	0xB80E56761ea92C5006848c34fE2401D151eC3Fe9	0x7c74006e39a0ceeb47181
Ecosystem	0x2827B43bfc104bd40F21355643ef7C5b8ea602ee	0x20b0d01b5b982e962354
Liquidity	0xaFfd89cf0D92A7AC9EaA4325102E5d2c5815d071	0xbd3131bf8b97e32cbacad

WOC-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	**Major**	**WorldOfDypians.sol: 701**	**Mitigated**

Description

In new commit `bcf3c5fa7729f1f59aa9fbf42288f7a36362088e` :

All of the `WoD` tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the anonymous deployer can distribute tokens without obtaining the consensus of the community. Any compromise to the deployer account that holds undistributed tokens may allow the attacker to steal and sell tokens on the market, resulting in severe damage to the project.

```
704     constructor(uint256 MAX_SUPPLY_TOKEN) Ownable(msg.sender) {
705         MAX_SUPPLY = MAX_SUPPLY_TOKEN;
706         _mint(msg.sender, MAX_SUPPLY_TOKEN);
707     }
```

Recommendation

It's recommended the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team shall make enough efforts to restrict the access of the private key. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to the private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize project teams with a third-party KYC provider to create greater accountability.

Alleviation

[World of Dypians Team, 11/27/2024]: We have distributed the tokens accordingly with our tokenomics. Here you can find our public tokenomics:

https://www.worldofdypians.com/tokenomics

token address: https://bscscan.com/address/0xb994882a1b9bd98a71dd6ea5f61577c42848b0e8

The total token supply is 1,000,000,000 $WOD

These are the Vesting Wallets that we have used accordingly with our tokenomics:

CATEGORY	ADDRESS	TOKENS
Seed	0xD62FC589701C1FC54675124C42fe1D7fF4e0204C	78,000,000 $WOD

CATEGORY	ADDRESS	TOKENS
Private	0x0A3C5eE8F6F7b552E436f922e4F3a28E24343f7b	83,219,784 $WOD
Private-IDO	0x9f149D2d422a12Ba34bee11473863625B9793B66	1,010,985 $WOD
KOL	0xaD07ef12F836409FF0d7206860Fd0174F7Bda342	12,973,855 $WOD
Team	0x218b135F64669cb102F0810A49E64E437C85c9F2	120,000,000 $WOD
Advisors	0x255b1C2e3f2FF180d45f1e055224d97b23079513	50,000,000 $WOD
Community	0xB80E56761ea92C5006848c34fE2401D151eC3Fe9	300,000,000 $WOD
Ecosystem	0x2827B43bfc104bd40F21355643ef7C5b8ea602ee	250,000,000 $WOD
Liquidity	0xaFfd89cf0D92A7AC9EaA4325102E5d2c5815d071	80,000,000 $WOD

WOC-01 | NO PUBLIC MINT FUNCTION

Category	Severity	Location	Status
Logical Issue	Medium	WorldOfDypians.sol: 701	Resolved

Description

The contract lacks a mint function with either a `public` or `external` modifier, making it impossible to mint tokens.

Recommendation

We recommend creating a mint function with the appropriate `public` or `external` visibility modifier.

Alleviation

[World of Dypians Team, 06/21/2024]: The team heeded the advice and resolved the issue in commit: bcf3c5fa7729f1f59aa9fbf42288f7a36362088e.

VCW-03 | MISSING CHECKS ON FUNCTION `addVestingWallets`

Category	Severity	Location	Status
Volatile Code	Minor	Vesting.sol: 931	Partially Resolved

Description

The function `addVestingWallets` adds wallet addresses and allocates tokens for vesting but misses input validation:

- It does not ensure that the `_holder` and `_amountToClaim` arrays are of equal length.
- It fails to check that the addresses in `_holder` are unique.
- It does not verify that the total of `_amountToClaim` does not exceed the contract's token balance.

These shortcomings could lead to invalid or duplicate addresses being added or more tokens being allocated than the contract holds, causing potential logical issues.

```
931     function addVestingWallets(address[] calldata _holders,
932                                uint[] calldata _amountToClaim)
933     external onlyOwner {
```

Recommendation

Revise the `addVestingWallets` function to include a preliminary validation step that ensures the `_holders` and `_amountToClaim` arrays are of equal length. Additionally, implement a mechanism to detect and prevent the inclusion of duplicate wallet addresses in `_holders`.

Alleviation

[CertiK, 06/27/2024]: The team heeded the advice and partially resolved the issue in commit: c76dca355862df4d42f350bd6cc72eb416782eb2.

The following issue still exists:

```
- It fails to check that the addresses in `_holder` are unique.
- It does not verify that the total of `_amountToClaim` does not exceed the
contract's token balance.
```

WOC-04 PULL-OVER-PUSH PATTERN IN `transferOwnership()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	WorldOfDypians.sol: 691~696	Acknowledged

Description

The change of `_owner` by function `transferOwnership()` overrides the previously set `_owner` with the new one without guaranteeing the new `_owner` is able to actuate transactions on-chain.

Recommendation

We advise the pull-over-push pattern to be applied here whereby a new `owner` is first proposed and consequently needs to accept the `_owner` status ensuring that the account can actuate transactions on-chain. The following code snippet can be taken as a reference:

```
address public potentialOwner;

function transferOwnership(address pendingOwner) external onlyOwner {
    require(pendingOwner != address(0), "potential owner can not be the zero
address.")
    potentialOwner = pendingOwner;
    emit OwnerNominated(pendingOwner);
}

function acceptOwnership() external {
    require(msg.sender == potentialOwner, 'You must be nominated as potential owner
before you can accept ownership');
    emit OwnerChanged(_owner, potentialOwner);
    _owner = potentialOwner;
    potentialOwner = address(0);
}
```

Alleviation

[World of Dypians Team, 06/21/2024]: The team acknowledged the finding and decided not to change the current codebase.

OPTIMIZATIONS | WORLD OF DYPIANS - AUDIT

ID	Title	Category	Severity	Status
VCW-01	State Variable Should Be Declared Constant	Coding Issue	Optimization	Resolved

VCW-01 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Coding Issue	Optimization	Vesting.sol: 826, 838	Resolved

Description

State variables that never change should be declared as `constant` to save gas.

```
826     uint public lockDuration = 30 minutes;
```

- `lockDuration` should be declared `constant`.

```
838     uint public cliff = 10 minutes;
```

- `cliff` should be declared `constant`.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

Alleviation

[World of Dypians Team, 06/21/2024]: The team heeded the advice and resolved the issue in commit: 0100f5e94db1740f30c8b52bacc0e70f32f308dd.

FORMAL VERIFICATION | WORLD OF DYPIANS - AUDIT

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
ownable-owner-succeed-normal	`owner` Always Succeeds
ownable-transferownership-correct	Ownership is Transferred
ownable-renounceownership-correct	Ownership is Removed
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows (note that overflow properties were excluded from the verification):

Property Name	Title
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed

Property Name	Title
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-approve-never-return-false	`approve` Never Returns `false`

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract TokenVestingLock (Contracts/Vesting.sol) In Commit 828217ac151d57fd3cb73350a8c3a4c72b6b91ea

Verification of Standard Ownable Properties

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	● True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	● True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	● True	
ownable-renounce-ownership-is-permanent	● True	

Detailed Results For Contract WorldOfDypians (Contracts/WorldOfDypians.sol) In Commit 828217ac151d57fd3cb73350a8c3a4c72b6b91ea

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-correct-amount	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-false	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-never-return-false	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-change-state	True	
erc20-totalsupply-correct-value	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-correct-value	True	
erc20-balanceof-succeed-always	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-correct-value	True	
erc20-allowance-succeed-always	True	
erc20-allowance-change-state	True	

APPENDIX | WORLD OF DYPIANS - AUDIT

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean

connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `WorldOfDypians` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                 && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract WorldOfDypians must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract WorldOfDypians.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Description of the Analyzed Ownable Properties

Properties related to function `owner`

ownable-owner-succeed-normal

Function `owner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.