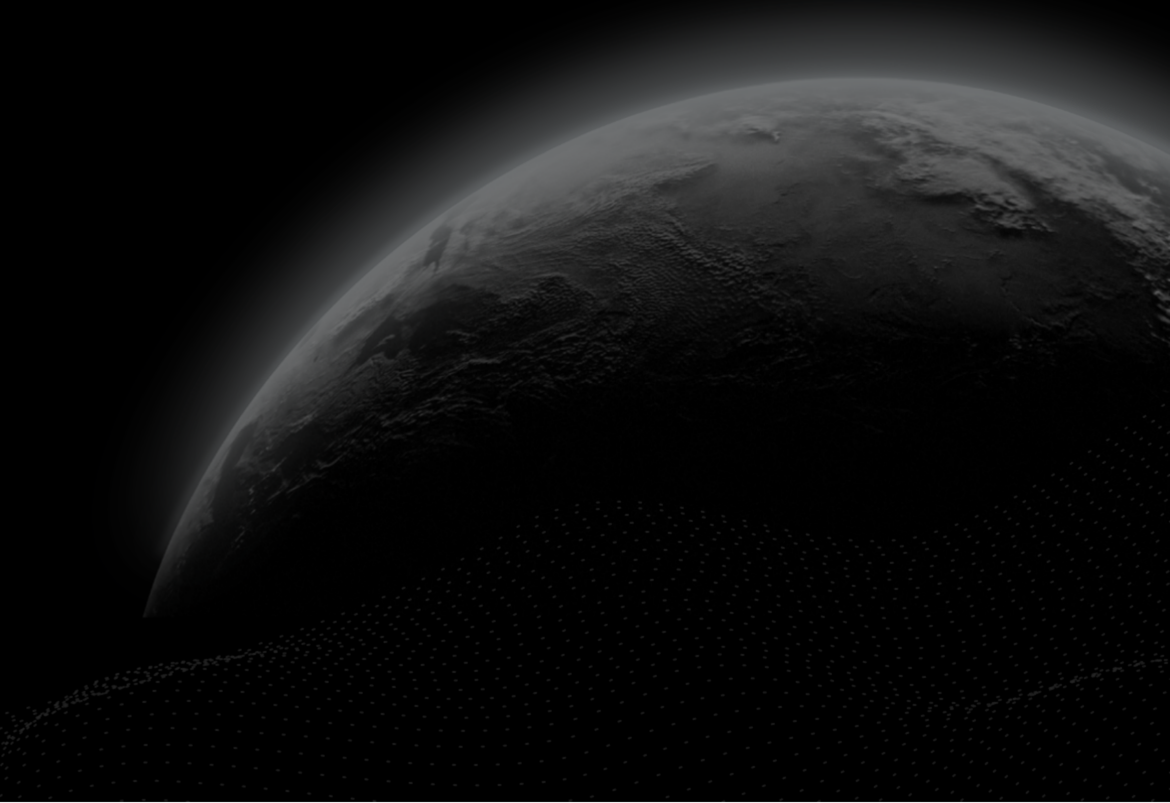
Security Assessment

Xdoge

CertiK Assessed on Sept 18th, 2023



CertiK Assessed on Sept 18th, 2023

Xdoge

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending	Arbitrum (ARB)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 09/18/2023	N/A

CODEBASE

xdoge-token futures-contract

View All in Codebase Page

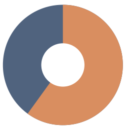
COMMITS

f1ac9aef556bb4fc01d81e002e1257f4abfacc9c

a87e412c40f4134612874d2361495b1026620121

View All in Codebase Page

Vulnerability Summary



5	0	0	0	5	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.
🟧	3	Major	3 Acknowledged	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.
🟨	0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	0	Minor		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.
🟦	2	Informational	2 Acknowledged	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 | XDOGE

CODEBASE | XDOGE

Repository

xdoge-token futures-contract

Commit

f1ac9aef556bb4fc01d81e002e1257f4abfacc9c a87e412c40f4134612874d2361495b1026620121

AUDIT SCOPE | XDOGE

84 files audited ● 9 files with Acknowledged findings ● 75 files without findings

ID	File	SHA256 Checksum
● BMF	diamond/facets/BrokerManagerFacet.sol	e4277e00dc5b969e3a5a1deb8b3c5df3a8002eeb622008c8dc359388959e2355
● FMF	diamond/facets/FeeManagerFacet.sol	83a5c7989812be359d519c70b9547513501d95dc97d157d2d46f7e91f1b22ec1
● PFF	diamond/facets/PriceFacadeFacet.sol	6fd8e68990320e27f0d96a54afb134cc8ce8b451c4494cfa751b4b9270552fe5
● SRF	diamond/facets/StakeRewardFacet.sol	4094bed5a687f755504381d834d5db2e751e2eeaae150b6e57f06ca9f95d0d03
● TRF	diamond/facets/TokenRewardFacet.sol	30c24dccea7aa7805ef8902579e8d2695a7cfc6c0785fefe2765801dc5e2c54a
● TPF	diamond/facets/TradingPortalFacet.sol	eb0dec95d2018d91b8316825cdd50b6dfbfc612667788ca89bdd766aca3bb54d
● LDB	diamond/libraries/LibDiamond.sol	e0e9371dfa4829cbb48bd99805637adce306930c5eb688064aa6706891673473
● LPF	diamond/libraries/LibPriceFacade.sol	a7769669ed36083fe4921549d535ad709b9eb76f9fdcfe3aa123b5aa1d0c5e25
● LTC	diamond/libraries/LibTradingConfig.sol	ea6bb7960fa9c88886c26e8d6c224b266a99eca63d55b0cc37afd2b5baba0da5
● IWE	dependencies/IWETH.sol	f36947072eef0f345a39dae3e0ada48bdf49971c171cefd6563e266b8200d029
● ACE	diamond/facets/AccessControlEnumerableFacet.sol	8d3f00cc03713fdb7bc18d462390da8c4f1e6a4743a6871beeb0c9140e2c7648
● CPF	diamond/facets/ChainlinkPriceFacet.sol	b04c67105121f9139a1048c19816ec2c7ab78c8b01361109e0ef30dba9e927d4
● DCF	diamond/facets/DiamondCutFacet.sol	6754977d5831c0bad40ae4237816914f371eb070e3388d93364872bcb8d05c38

ID	File	SHA256 Checksum
DLF	diamond/facets/DiamondLoupeFacet.sol	87cd7272e0c67287d13b7aea815b0e030bbe4de39bfa061739cfcb243d7e2fc0
LOF	diamond/facets/LimitOrderFacet.sol	f3b41951a07ec1a35f9bd030e3e95357e8ca6380f3db93fcddb267878c8e56eb
LMF	diamond/facets/LpManagerFacet.sol	45f607b1efcfba64e03f051ddfe7c12addfb6234dd23c6ace3f194f316e6b2c2
OAT	diamond/facets/OrderAndTradeHistoryFacet.sol	64a6ec7775375410c60aaa3b51fcdb4aaf0708b755c172ee8307e968fedbb2c3
PMF	diamond/facets/PairsManagerFacet.sol	8c2c9bcf4a6d36d30e58620c0e1b6b49bf6d3351f1fba997009ca379624eb413
PFB	diamond/facets/PausableFacet.sol	65d512cf886a7da9b8a7d0cdcab302bbf4d94c5996756a47536389ba00fb622e
TLF	diamond/facets/TimeLockFacet.sol	347e997ef6eaaafe22afef9a20ad553ab3940c7dc683656812f1ac6d99abdb77
TCF	diamond/facets/TradingCheckerFacet.sol	2a7541e3ca0b4f814e1c6bb225b9cdbbe4ecad3b77e79d2572169e01d9701a49
TRA	diamond/facets/TradingCloseFacet.sol	2f08825708b3ae0f8b4ec9c8d227a78c19fc3522251198e198fdfc19ab241ae7
TRD	diamond/facets/TradingConfigFacet.sol	c2a8c3f76dc637c491e064492b1537ca679e0f137ef77f32aa42716763a0fc1f
TRI	diamond/facets/TradingCoreFacet.sol	64cc85bd1ebc102584f305e4a35a5f087f98f1437d5e055517e49a126d811918
TOF	diamond/facets/TradingOpenFacet.sol	91f1aaa4a929b9c02319f76b974b30204976f8f3251ef2293338b29538c5793e
TRN	diamond/facets/TradingReaderFacet.sol	5bcc375616039b8cfa8389f23436ca91dfa3ff6eb14f24416ca70c8d82a9a15a
TFB	diamond/facets/TransitionFacet.sol	12c4b74a5e9e073364830650905ee9bef602683711f2a400897690146d5cfbf2
VFB	diamond/facets/VaultFacet.sol	427f49c18cf193cb49f6a80aacd558221cd12c6cfb929c4dc0286742de0ebe8b
IAS	diamond/interfaces/IArbSys.sol	0933fb0447ebf1f9c11705d72190a12449e14ea372b6f5327ffdfb4c52c1bd66

ID	File	SHA256 Checksum
IBB	diamond/interfaces/IBook.sol	99d0d9bf1d35c4554386e0373876eabb34dbd32f97e239832f0697b5ea582fda
IBM	diamond/interfaces/IBrokerManager.sol	a9a3c5025b5645da048c6eedd92b7c752e3c1542c7f077d779d4036531e7f65b
ICP	diamond/interfaces/IChainlinkPrice.sol	921e9282452aa4c9355a27dee94c3775306310368459615cef8bd98954f59f86
IDC	diamond/interfaces/IDiamondCut.sol	97d6b3c39de92dfdee49ef911745afc49c95af2d611db1f1ff1a1c1bb0705b2d
IDL	diamond/interfaces/IDiamondLoupe.sol	b5687b75080a1d5d76215b6981e89a2385479696d372bccd9a99cc88fc5a1cc4
IFM	diamond/interfaces/IFeeManager.sol	0c10a30a83a093d6d3b7f1d38f78b403beb065eb83606424f815b83e3e9d370e
ILO	diamond/interfaces/ILimitOrder.sol	ca4e76abd71e903a10842f19cfb12a8149c4da760b1489106f716b4bd7e33b2b
ILB	diamond/interfaces/ILp.sol	a4132bceb35999faf3e28b982fe9b55dd1d87189bc516ecd5d9d9d1b52cb9b0f
ILM	diamond/interfaces/ILpManager.sol	db7cefe39c438420942cc1d0a9d76875761ab0f8f3f44b71b03286da70f2c9f1
IOP	diamond/interfaces/IOraclePrice.sol	ca14fecc11802ffec0026b193f81661b4335f7cdc77737169d37f70c26cf5300
IOA	diamond/interfaces/IOrderAndTradeHistory.sol	186dd05eb6669ff32fe7c627720ee92821f3d864835c7bf93b170c770b143ec1
IPM	diamond/interfaces/IPairsManager.sol	89b741894b7dccc0b9925a6e960da549250eba9599c4072b197760f8c78a4a71
IPB	diamond/interfaces/IPausable.sol	6ee03369f716f034140057c3088e215b3f35d98b2c4e6fd41b00d9c90458221a
IPF	diamond/interfaces/IPriceFacade.sol	e9d058a810cd72b7e0b6a9b0a9e478182abc052ad4714d4374cadb3c1f52a095
ISR	diamond/interfaces/IStakeReward.sol	c487083d0e93802fbcfa4808a79d827f0548c33d7836ecf468980a238aef1e68
ITL	diamond/interfaces/ITimeLock.sol	0191f070a5676e5d9d69ea5835d87f7f1b936a1d0c57446aa6d3a101e8775306

ID	File	SHA256 Checksum
● ITR	diamond/interfaces/ITokenReward.sol	d64662ee540599cc6fa0a0bea1d2a820db6de138bf141f5e552a0380854f00ae
● ITB	diamond/interfaces/ITrading.sol	f8305e0430b11229a2255713c89587a7885bc90a9a6de1d80b8add09935801b9
● ITC	diamond/interfaces/ITradingChecker.sol	7e9d8d0bae225efc364784c458e03aa88fdfcfa2e3b48ed1d38e80c459436f13
● ITA	diamond/interfaces/ITradingClose.sol	4687661f25321df1af970249c57d82a6791176c01d72e884a8aed3e138b9495d
● ITD	diamond/interfaces/ITradingConfig.sol	7c65380f2d750f29b76722e9181376e330e07253c90f0a6b7a7c01ad635401d2
● ITI	diamond/interfaces/ITradingCore.sol	863125ab802d2711f74595d523e0a9784cca6785b70690e25ff7dc377c8f31e9
● ITO	diamond/interfaces/ITradingOpen.sol	b1cd9e6481ecd710e2161bd171134c0d9973dd77ec8462615dbe9fdfe988ff9d
● ITP	diamond/interfaces/ITradingPortal.sol	1c0eb23075cca9ce5178782cfade671790d5817cfa6f19d0e6b396e4bf682979
● ITN	diamond/interfaces/ITradingReader.sol	2e38b3ea354aca837c717eb011277833d7f091724286a521a5856fe71e2231a0
● IVB	diamond/interfaces/IVault.sol	b2ca8004f30a6c4a3d4f5a7a1b1009c3037bd526ee87aa72d152d1effe390ec7
● LAC	diamond/libraries/LibAccessControlEnumerable.sol	ba8c1d14ba3420d420040f6795e18a7503dd6070bd7f66ce5f6eea816a713658
● LBM	diamond/libraries/LibBrokerManager.sol	76db225c2a7113b269d1778f9139a6448c6ee7cad962699a12e4cdf40837ff4e
● LCB	diamond/libraries/LibChain.sol	ef95392fb487db7780e7d0d85929f41c5db436e0f8b4e76a8e1fec542de2673b
● LCP	diamond/libraries/LibChainlinkPrice.sol	e86d66b7b11b7824b7aa76438bcf9dc59d35d72ad767f72fca9b5339759fab92
● LFM	diamond/libraries/LibFeeManager.sol	639f378212366f60b0dca14438ec3e8b0ce6300a54007df80bdd267a44727875

ID	File	SHA256 Checksum
● LLO	diamond/libraries/LibLimitOrder.sol	6ed9ad3df9b60cfd7210d01dc1a7eec6977042d0fb0c5167527463214ce062a3
● LLM	diamond/libraries/LibLpManager.sol	942b7ebc9bfd9ecf16c0e521d41057e685ca0976ed74c4a444ebddf6360a2724
● LOA	diamond/libraries/LibOrderAndTradeHistory.sol	3cef0f3d80214c84e98b5c3f4840a6967493c3747d5c42f5e7039758f38ce004
● LPM	diamond/libraries/LibPairsManager.sol	26387004a5c3fb90c85b2975ce0c60975629758523eb120e018ffbbc7fe58112
● LSR	diamond/libraries/LibStakeReward.sol	8c58b367839243524f3fd3365fbb6d8d59441bbddf344bb42fcdee21537be57c
● LTL	diamond/libraries/LibTimeLock.sol	1b8c1a395dc79cfad1487eaee01146fb1584a130370024766d24e99c7e81f638
● LTR	diamond/libraries/LibTokenReward.sol	9b761aea12c713e1987ebfb52fd16b714018a7b03cdf2b5fc4230ffbfe825ed3
● LTB	diamond/libraries/LibTrading.sol	ade7bbb898c3d65ee54c42d821c55564290012d0f0c1873233ab98ab619fef74
● LIT	diamond/libraries/LibTradingCore.sol	2e0f04decc72df4a9c516b6cf1a8f5160d6029042048a0adc87ebedc9c756d5c
● LVB	diamond/libraries/LibVault.sol	47556f43e2d00bd4013c91355ced970866e103ec5f887f6425146fff4b5650bb
● OSB	diamond/security/OnlySelf.sol	b6ef5a9bcf8b4a0847f92765efa69d089638d1b994719bf171a8a331cc94a26b
● PAU	diamond/security/Pausable.sol	cc58847253426780f227848b0aacc292f0a8fcb1a1005900431d8de0dbb279f7
● RGB	diamond/security/ReentrancyGuard.sol	703983195aab202beda65f8281f2f46d8c81ebf58f2f38689783e23af11eddec
● TII	diamond/upgradeInitializers/TrademanInit.sol	e90dec7a2528be72e127add8a01a9638eaa7c15b4bf8ddc931167e7890ace8f0
● TXB	diamond/TradeX.sol	94224a205c1d159e26e86919ca234971ba394c5ecd2c2620265e261b1f46cbdd
● MDC	test/MockDiamondCutFacet.sol	04b340c95ea35da6d98799da5d1f243331c276ea54cb0a015575e005cf1b89f9

ID	File	SHA256 Checksum
MTA	test/MockTrademanAggregator.sol	aba6c5c52c7e7cd7a2c22ac128df937d4f48ce4ea34e910ec30affe2ffb3fc46
MTE	test/MockTrademanERC20.sol	b8b0f339ba53b30b85482c5fd8eae35eb48d354b18f68e6410ed374d1c714079
MTI	test/MockTrademanInit.sol	2275fa1f37ff5fe8c7e0573ea4f0703ccfd1ce9192d478cfeb55859292cce94b
MTW	test/MockTrademanWETH9.sol	166dedbc1949c2bd3358f301582539402117f844b7dced7f90dfc435a3d6a9b6
BIT	utils/Bits.sol	98b01bac7d4fb1e34651578762778241e7ca8d2dc845876e2171e8a832391074
COS	utils/Constants.sol	d92d9ed0189a5fa3c6578ae96759153fde28f6a0cc9fde5dd372dccbdab7b5a7
TXL	TradeXLP.sol	578467ed09307d077aebca6089c36bb9f3d1525020be772d74d4af9800bab8cc
XDG	xdoge.sol	3456ef5006f7e5d73a97200e2a74555560dec299051b43616d9374d264c6fe9f

APPROACH & METHODS | XDOGE

This report has been prepared for Xdoge to discover issues and vulnerabilities in the source code of the Xdoge 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.

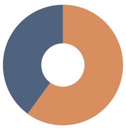
The auditing process pays special attention to the following considerations:

- 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 | XDOGE



5	0	3	0	0	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Xdoge. 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:

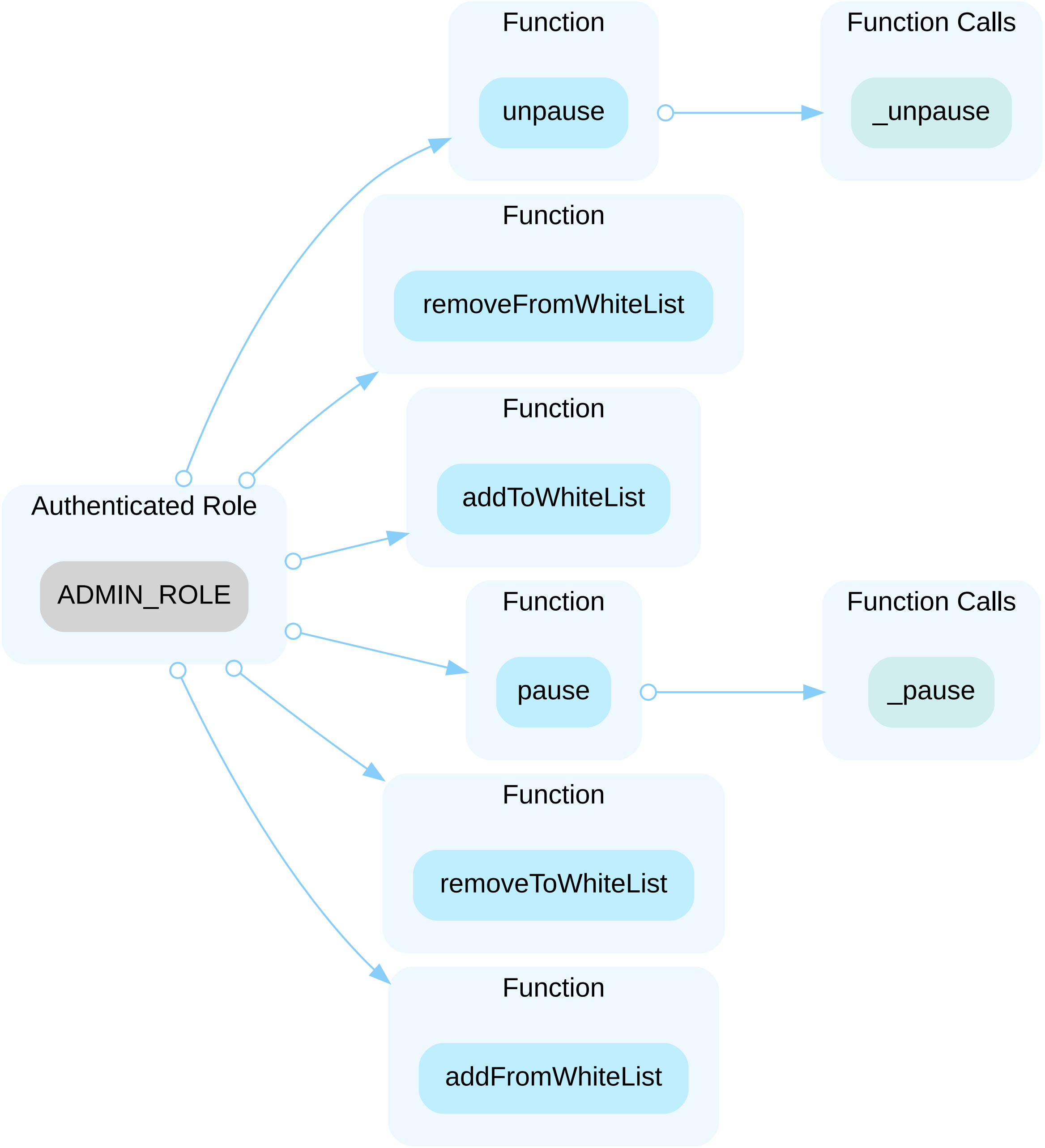
ID	Title	Category	Severity	Status
GLOBAL-01	**Centralization Risks In TradeXLP.Sol**	**Centralization**	**Major**	● **Acknowledged**
GLOBAL-02	**Centralization Risks In TradeX.Sol**	**Centralization**	**Major**	● **Acknowledged**
XDG-02	**Initial Token Distribution**	**Centralization**	**Major**	● **Acknowledged**
DIA-01	Inaccurate Error Message	Coding Style	Informational	● Acknowledged
LPF-01	Inaccurate Comment	Coding Style	Informational	● Acknowledged

GLOBAL-01 | CENTRALIZATION RISKS IN TRADEXLP.SOL

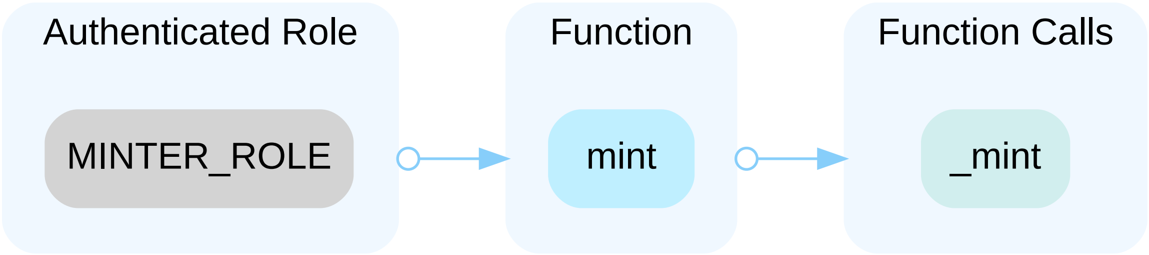
Category	Severity	Location	Status
Centralization	● Major		● Acknowledged

Description

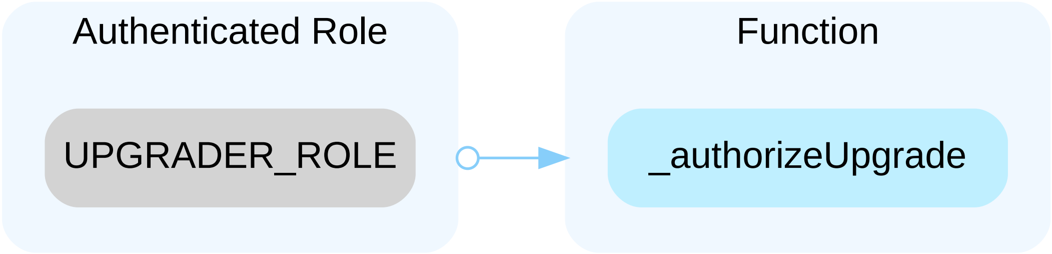
In the contract `TradeXLP` the role `ADMIN_ROLE` has authority over the functions shown in the diagram below.



In the contract `TradeXLP` the role `MINTER_ROLE` has authority over the functions shown in the diagram below.



In the contract `TradeXLP` the role `UPGRADER_ROLE` has authority over the functions shown in the diagram below.



Any compromise to the privileged roles may allow the hacker to take advantage of this and

- `mint()` any amount of `TradeXLP`
- `upgradeTo()` any other implementation contract
- `pause()` / `unpause()` , update whitelists

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at 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 failure.

- 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.

GLOBAL-02 | CENTRALIZATION RISKS IN TRADEX.SOL

Category	Severity	Location	Status
Centralization	● Major		● Acknowledged

Description

In the contract `TradeX`

- the role `DEPLOYER_ROLE` has the authority to upgrade all facets and initialize them.
- the role `DEFAULT_ADMIN_ROLE` has the authority to edit other roles.
- other roles can perform sensitive operations.

Any compromise to the privileged roles may allow the hacker to take advantage of this and

- upgrade any facet with new functionality
- add/remove pairs/brokers/commissions, etc.
- update staking reward via `updateTokenPerBlock()`
- provide any prices and execute the orders

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at 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 failure.

- 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.

XDG-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● **Major**	**xdoge.sol (token): 20**	● **Acknowledged**

Description

All of the `SimpleToken` tokens are sent to the contract deployer. This is a centralization risk because the deployer can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

It is recommended that 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 should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a 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 the project team with a third-party KYC provider to create greater accountability.

DIA-01 | INACCURATE ERROR MESSAGE

Category	Severity	Location	Status
Coding Style	● Informational	diamond/facets/BrokerManagerFacet.sol (perpetual): 17; diamond/facets/FeeManagerFacet.sol (perpetual): 17; diamond/facets/PriceFacadeFacet.sol (perpetual): 27; diamond/facets/StakeRewardFacet.sol (perpetual): 16; diamond/facets/TokenRewardFacet.sol (perpetual): 13; diamond/facets/TradingPortalFacet.sol (perpetual): 59, 139; diamond/libraries/LibDiamond.sol (perpetual): 173; diamond/libraries/LibPriceFacade.sol (perpetual): 90, 135; diamond/libraries/LibTradingConfig.sol (perpetual): 59, 67	● Acknowledged

Description

```
173         enforceHasContractCode(_init, "LibDiamondCut: Init address has no code"
);
```

"Init" is supposed to be "_init".

```
90              require(highPriceGapP > lowPriceGapP,
"LibPriceFacade: HighPriceGapP must be greater than lowPriceGapP");
```

"HighPriceGapP" is supposed to be "highPriceGapP".

Recommendation

We recommend updating the error messages.

LPF-01 | INACCURATE COMMENT

Category	Severity	Location	Status
Coding Style	● Informational	diamond/libraries/LibPriceFacade.sol (perpetual): 37	● Acknowledged

Description

```
37          // keccak256(token, block.number) =>
```

`block.number` is supposed to be `LibChain.getBlockNumber()` .

Recommendation

We recommend updating the comment.

OPTIMIZATIONS | XDOGE

ID	Title	Category	Severity	Status
XDG-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Acknowledged

XDG-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	xdoge.sol (token): 9	● Acknowledged

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as `immutable`.

FORMAL VERIFICATION | XDOGE

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 automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

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 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:

Property Name	Title
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transfer-false	If `transfer` Returns `false` , the Contract State Is Not Changed
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address

Property Name	Title
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-transferfrom-false	If `transferFrom` Returns `false` , the Contract's State Is Unchanged
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State

Property Name	Title
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-false	If `approve` Returns `false` , the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`

Verification Results

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
  - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
  - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.
- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
  - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
  - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract TradeXLP (contracts/TradeXLP.sol) In Commit a87e412c40f4134612874d2361495b1026620121

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-succeed-normal	● False	
erc20-transfer-succeed-self	● False	
erc20-transfer-exceed-balance	● True	
erc20-transfer-change-state	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	
erc20-transfer-recipient-overflow	● False	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-succeed-normal	● False	
erc20-transferfrom-succeed-self	● False	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-fail-recipient-overflow	● False	

Detailed results for function `totalSupply`

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

Detailed results for function `balanceOf`

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

Detailed results for function `allowance`

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

Detailed results for function `approve`

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

Detailed Results For Contract MockTrademanERC20 (contracts/test/MockTrademanERC20.sol) In Commit a87e412c40f4134612874d2361495b1026620121

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-succeed-self	● True	
erc20-transfer-correct-amount	● True	
erc20-transfer-change-state	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-exceed-balance	● True	
erc20-transfer-false	● True	
erc20-transfer-never-return-false	● True	
erc20-transfer-recipient-overflow	● False	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● True	
erc20-transferfrom-revert-to-zero	● True	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-change-state	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-fail-recipient-overflow	● False	

Detailed results for function `totalSupply`

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

Detailed results for function `balanceOf`

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

Detailed results for function `allowance`

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

Detailed results for function `approve`

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

Detailed Results For Contract MockTrademanWETH9 (contracts/test/MockTrademanWETH9.sol) In Commit a87e412c40f4134612874d2361495b1026620121

Verification of ERC-20 Compliance

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-succeed-self	● True	
erc20-transfer-succeed-normal	● True	
erc20-transfer-change-state	● Inapplicable	
erc20-transfer-correct-amount	● True	
erc20-transfer-revert-zero	● False	
erc20-transfer-false	● Inapplicable	
erc20-transfer-exceed-balance	● True	
erc20-transfer-correct-amount-self	● True	
erc20-transfer-recipient-overflow	● True	
erc20-transfer-never-return-false	● True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	● False	
erc20-transferfrom-succeed-normal	● True	
erc20-transferfrom-revert-to-zero	● False	
erc20-transferfrom-succeed-self	● True	
erc20-transferfrom-change-state	● Inapplicable	
erc20-transferfrom-correct-amount-self	● True	
erc20-transferfrom-correct-amount	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-false	● Inapplicable	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-fail-recipient-overflow	● True	
erc20-transferfrom-never-return-false	● True	

Detailed results for function `totalSupply`

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

Detailed results for function `balanceOf`

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

Detailed results for function `allowance`

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

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-change-state	● Inapplicable	
erc20-approve-false	● Inapplicable	
erc20-approve-succeed-normal	● True	
erc20-approve-correct-amount	● True	
erc20-approve-never-return-false	● True	
erc20-approve-revert-zero	● False	

APPENDIX | XDOGE

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
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

Technical description

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which 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 model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.

- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to an ERC-20 token 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 definitions

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time steps. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

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:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` .
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond` . Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond` .

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer` , `transferFrom` , `approve` , `allowance` , `balanceOf` , and `totalSupply` .

In the following, we list those property specifications.

Properties for ERC-20 function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address.

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

Specification:

```
[](started(contract.transfer(to, value), to == address(0))
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers.

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to != msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[to] + value <= type(uint256).max && _balances[to] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers.

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to == msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[msg.sender] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers.

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:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender
   && _balances[to] >= 0 && value >= 0
   && _balances[to] + value <= type(uint256).max
   && _balances[msg.sender] >= 0 && _balances[msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transfer(to, value), return
             ==> _balances[msg.sender] == old(_balances[msg.sender]) - value
                 && _balances[to] == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender
  && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
     ==> <>(finished(contract.transfer(to, value), return
         ==> _balances[to] == old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to)
  ==> <>(finished(contract.transfer(to, value), return
      ==> (_totalSupply == old(_totalSupply) && _allowances == old(_allowances)
           && _balances[p1] == old(_balances[p1])  ))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance.

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

Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender]
   && _balances[msg.sender] >= 0 && value <= type(uint256).max)
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance.

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
[](started(contract.transfer(to, value), to != msg.sender
   && _balances[to] + value > type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max
   && _balances[msg.sender] <= type(uint256).max
   && value > 0 && value <= _balances[msg.sender])
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return) || finished(contract.transfer(to, value), _balances[to]
                      > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transfer-false

If Function `transfer` Returns `false`, the Contract State Has Not Been Changed.

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

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return]
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transfer-never-return-false

Function `transfe` Never Returns `false`.

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

Specification:

```
[](!(finished(contract.transfer, !return)))
```

Properties for ERC-20 function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && to != address(0) && from != to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && _balances[to] + value <= type(uint256).max
   && value >= 0 && _balances[to] >= 0 && _balances[from] >= 0
   && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers.

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` , and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && from == to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && value >= 0 && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] <= type(uint256).max)
     ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers.

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:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0
&& _balances[from] >= 0 && _balances[from] <= type(uint256).max
&& _balances[to] >= 0 && _balances[to] + value <= type(uint256).max)
   ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> _balances[from] == old(_balances[from]) - value
              && _balances[to] == old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to
   && value >= 0 && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
             ==> _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly.

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:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0
   && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max && _balances[to] >= 0
   && _balances[to] <= type(uint256).max && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> ((_allowances[from][msg.sender]
                == old(_allowances[from][msg.sender]) - value)
                 || (_allowances[from][msg.sender]
                     == old(_allowances[from][msg.sender])
                        && (from == msg.sender
                             || old(_allowances[from][msg.sender])
                                == type(uint256).max))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest` ,
- The balance entry for the address in `from` ,

- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to
   && (p2 != from || p3 != msg.sender))
    ==> <>(finished(contract.transferFrom(from, to, amount), return
     ==> (_totalSupply == old(_totalSupply) && _balances[p1] == old(_balances[p1])
         && _allowances[p2][p3] == old(_allowances[p2][p3])  ))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available 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:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from]
   && _balances[from] >= 0 && _balances[from] <= type(uint256).max)
       ==> <>(reverted(contract.transferFrom)
              || finished(contract.transferFrom, !return)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available 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:

```
  [](started(contract.transferFrom(from, to, value), value > _allowances[from]
[msg.sender]
     && _allowances[from][msg.sender] >= 0 && value <= type(uint256).max)
     ==> <>(reverted(contract.transferFrom)
           || finished(contract.transferFrom(from, to, value), !return)
           || finished(contract.transferFrom(from, to, value), return
              && (msg.sender == from
                  || _allowances[from][msg.sender] == type(uint256).max))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance.

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != to
   && _balances[to] + value > type(uint256).max && value <= type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
      ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), _balances[to]
              > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed.

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

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`.

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

Specification:

```
[](!(finished(contract.transferFrom, !return)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds.

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

Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable.

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

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, return == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State.

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

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, _totalSupply == old(_totalSupply)
          && _balances == old(_balances) && _allowances == old(_allowances)  )))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds.

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

Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value.

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

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State.

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

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), _totalSupply == old(_totalSupply)
          && _balances == old(_balances)
          && _allowances == old(_allowances)  )))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds.

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

Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value.

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

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
           return == _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State.

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

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
   _totalSupply == old(_totalSupply) && _balances == old(_balances)
                && _allowances == old(_allowances)  )))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address.

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

Specification:

```
[](started(contract.approve(spender, value), spender == address(0))
   ==> <>(reverted(contract.approve)
          || finished(contract.approve(spender, value), !return)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs.

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:

```
[](started(contract.approve(spender, value), spender != address(0))
   ==> <>(finished(contract.approve(spender, value), return)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly.

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:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && value >= 0 && value <= type(uint256).max)
       ==> <>(finished(contract.approve(spender, value), return
              ==> _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes.

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && (p1 != msg.sender || p2 != spender))
       ==> <>(finished(contract.approve(spender, value), return
              ==> _totalSupply == old(_totalSupply) && _balances == old(_balances)
                  && _allowances[p1][p2] == old(_allowances[p1][p2])  )))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed.

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

Specification:

```
[](willSucceed(contract.approve(spender, value))
   ==> <>(finished(contract.approve(spender, value), !return
          ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
               && _allowances == old(_allowances)  ))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false`.

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

Specification:

```
 [](!(finished(contract.approve, !return)))
```

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.

CertiK | Securing the Web3 World

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.