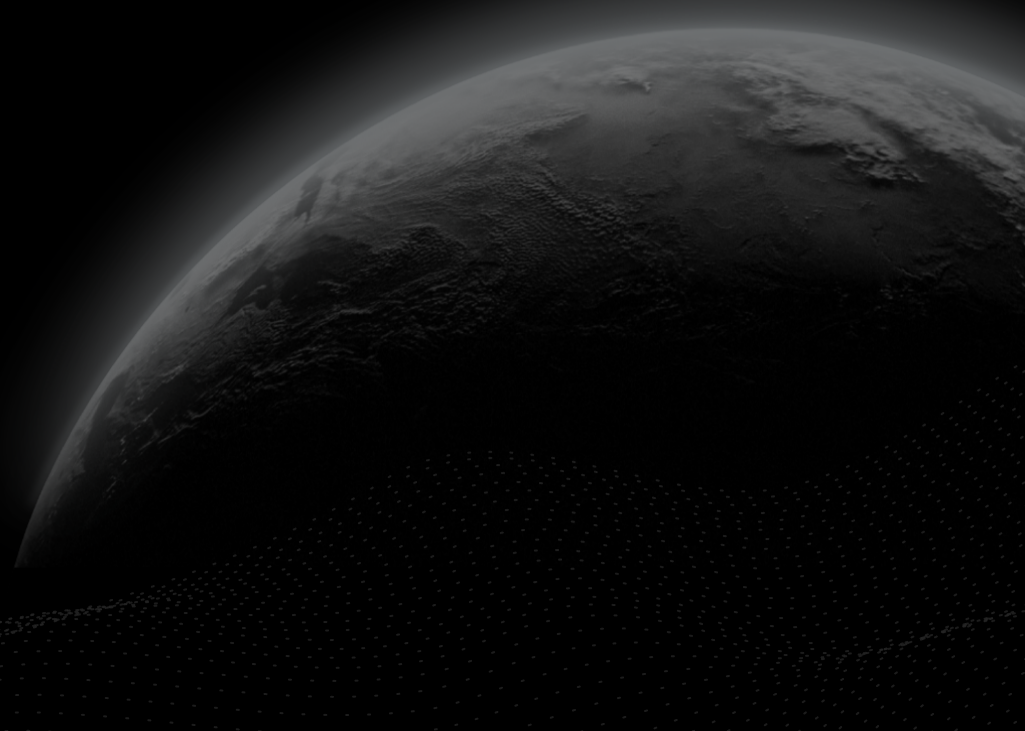
CERTIK

Security Assessment

XEN Crypto - Audit 2

CertiK Verified on Dec 27th, 2022



CertiK Verified on Dec 27th, 2022

XEN Crypto - Audit 2

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi, ERC-20	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
JavaScript, Solidity	Delivered on 12/27/2022	N/A

CODEBASE

https://github.com/FairCrypto/XENFT

...View All

Vulnerability Summary

8	4	0	0	4	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

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.
0	Major		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 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4	Minor	2 Resolved, 2 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.
3	Informational	2 Resolved, 1 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 | XEN CRYPTO - AUDIT 2

CODEBASE | XEN CRYPTO - AUDIT 2

Repository

https://github.com/FairCrypto/XENFT

AUDIT SCOPE | XEN CRYPTO - AUDIT 2

10 files audited ● 2 files with Acknowledged findings ● 8 files with Resolved findings

ID	File	SHA256 Checksum
● BPB	projects/XENFT-master/contracts/libs/BokkyPooBahsDateTimeLibrary.sol	44a69551064ec01633ba6152de485607cace11f3065d47c2b13d3319574e0d9d
● XET	projects/XENFT-master/contracts/XENFT.sol	ad66aaf757392d07016e4d0c7da6dfcddeffb9e233eaac005b0234e60e376d24
● AXE	projects/XENFT-master/contracts/libs/Array.sol	698b7f9b60251f93cc86422f1a3424301af389581e0c48376824918cb49cd1b8
● DTX	projects/XENFT-master/contracts/libs/DateTime.sol	44b9eb3072cdd960d652ad0d1b3bace3a59adfd15ceb20768ea99c31b527e104
● ERC	projects/XENFT-master/contracts/libs/ERC2771Context.sol	eeeab16a43fee256a592279314da5f1f0c728268e9976500cfccb604979e4349
● FSX	projects/XENFT-master/contracts/libs/FormattedStrings.sol	a3d2de38e7f569cd43120d98fea45e4c7cc68850a6de2d0263efa49dcaeafb0a
● MXE	projects/XENFT-master/contracts/libs/Metadata.sol	f5deeb704639c23465d62e9d8e90864c8171e1bfc58280df2b5c333ea2ac5f26
● MIX	projects/XENFT-master/contracts/libs/MintInfo.sol	2b5f5a9bd5d051220f92646f902dc1730e02f0c9e4615d008c3af81a69f1c597
● SVG	projects/XENFT-master/contracts/libs/SVG.sol	03fbb627ba27ed3fce84edd1a8ecf0575d0fa3ff5fb93c0dd4eae63588a1a1d5
● SDX	projects/XENFT-master/contracts/libs/StringData.sol	451864440b64c5c6847673c12d30a33ac4d58f895479e787a13cdf877b431cd8

APPROACH & METHODS | XEN CRYPTO - AUDIT 2

This report has been prepared for XEN Crypto to discover issues and vulnerabilities in the source code of the XEN Crypto - Audit 2 project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis 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 | XEN CRYPTO - AUDIT 2

8	0	0	1	4	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for XEN Crypto - Audit 2. Through this audit, we have uncovered 8 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
BPB-01	Divide Before Multiply	Mathematical Operations	Minor	Acknowledged
XET-01	Potential Revert Due To An Underflow In Function `_specialTier`	Mathematical Operations	Medium	Acknowledged
XET-02	Missing Zero Address Validation	Volatile Code	Minor	Resolved
XET-03	Potential Reentrancy Attack	Volatile Code	Minor	Resolved
GLOBAL-02	Question Related To Tokenomics	Logical Issue	Informational	Acknowledged
XEF-01	Different Solidity Versions	Language Specific	Informational	Resolved
XEF-02	Inconsistency Between Code And Documentation	Logical Issue	Informational	Resolved

GLOBAL-01 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor		Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party `XEN Crypto` protocol. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. Additionally, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of `XENFT` requires interaction with `XEN Crypto`. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[XEN Team]: Issue acknowledged. I won't make any changes for the current version.

That's correct, XEN Torrent is designed to be linked to original XEN Crypto contract which servers as on-chain market maker (it mints and burns XEN, which is the baseline for XEN Torrent XENFT contract).

XEN Crypto code is immutable, the contract is not upgradeable or changeable since its genesis. Linking it into XEN Torrent is also done at its genesis, making the link immutable. This is all by design, XEN rigorously follows the first principles of crypto, creating fair opportunities for everyone.

Such approach of course comes with a risk of a potential flaw that could break operations of XEN Crypto, XEN Torrent of both. However XEN community had a plenty of time to study the code base and test the contracts in the field for the prolonged periods of time before they were submitted to the audit

BPB-01 | DIVIDE BEFORE MULTIPLY

Category	Severity	Location	Status
Mathematical Operations	Minor	projects/XENFT-master/contracts/libs/BokkyPooBahsDateTimeLibrary.sol: 66~74, 108, 109, 110, 111, 112, 113, 114, 115	Acknowledged

Description

Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

```
66          int256 __days = _day -
67              32075 +
68              (1461 * (_year + 4800 + (_month - 14) / 12)) /
69              4 +
70              (367 * (_month - 2 - ((_month - 14) / 12) * 12)) /
71              12 -
72              (3 * ((_year + 4900 + (_month - 14) / 12) / 100)) /
73              4 -
74              _OFFSET19700101;
```

```
108         int256 N = (4 * L) / 146097;
```

```
109         L = L - (146097 * N + 3) / 4;
```

```
110         int256 _year = (4000 * (L + 1)) / 1461001;
```

```
111         L = L - (1461 * _year) / 4 + 31;
```

```
112         int256 _month = (80 * L) / 2447;
```

```
113         int256 _day = L - (2447 * _month) / 80;
```

```
114         L = _month / 11;
```

```
115         _month = _month + 2 - 12 * L;
```

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

[XEN Team]: Issue acknowledged. I won't make any changes for the current version. This is a 3rd party library with an extensive suite of tests. It's based on a battle-tested algorithms published by US military: https://aa.usno.navy.mil/faq/JD_formula.html The original algorithms place specific order of operations with truncation operations being among them, so we believe that it's safe to trust the results. In any case the usage of this library is purely decorative, to generate stringified data for NFT metadata art, so potential minor discrepancies are not damaging.

XET-01 POTENTIAL REVERT DUE TO AN UNDERFLOW IN FUNCTION `_specialTier`

Category	Severity	Location	Status
Mathematical Operations	Medium	projects/XENFT-master/contracts/XENFT.sol: 405~409, 467, 530	Acknowledged

Description

In the contract `XENFT`, function `_specialTier` is used to determine special class tier based on XEN to be burned.

Impact

If the array `specialSeriesBurnRates` is constructed as comments, specialSeriesBurnRates[1] is 0, and line 406 will revert. Furthermore, the function `bulkClaimRankLimited` would revert when minting `Limited` class tokens and no `Limited` tier tokens will ever be minted.

```
405         for (uint256 i = specialSeriesBurnRates.length - 1; i > 0; i--) {
406             if (burning > specialSeriesBurnRates[i] - 1) {
407                 return i;
408             }
409         }
```

Recommendation

We recommend the team to check the original design and perform corresponding changes.

Alleviation

[XEN Team]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/FairCrypto/XENFT/commit/419013286117de1fd3d65b364fe8c2b81ca37b73

additional checks for specialSeriesBurnRates[i] == 0 have been added (L443-445). Nevertheless, corresponding value for Limited category tokens will not be equal to zero, which will allow this tokens to be minted. This is shown in the test suite. Example values of specialSeriesBurnRates array are shown in the /config/genesisParams(.*).js files

Certik: If the burn rate of Limited NFT is 1 instead of 0, the above change is not necessary. The recommended comment change is already implemented in https://github.com/FairCrypto/XENFT/commit/419013286117de1fd3d65b364fe8c2b81ca37b73

XET-02 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	projects/XENFT-master/contracts/XENFT.sol: 364	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
364         _trustedForwarder = trustedForwarder;
```

- `trustedForwarder` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[XEN Team]: Issue acknowledged. I won't make any changes for the current version.

_trustedForwarder is used exclusively in the checking function function isTrustedForwarder(address forwarder) public view virtual returns (bool) { return forwarder == _trustedForwarder; } where it is compared to msg.sender. Therefore we don't see any risk associated with _trustedForwarder being equal to 0. In fact, setting of _trustedForwarder is optionally delayed from the moment of the contract instantiation, so it could be equal to 0 meaning that it's not yet set, which disables the functionality of a proxy transactions

XET-03 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Volatile Code	Minor	projects/XENFT-master/contracts/XENFT.sol: 280, 284, 284, 508	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

External call(s)

```
280         _safeMint(user, _tokenId);
```

- This function call executes the following external call(s).
- In `ERC721._checkOnERC721Received`,
  - `IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data)`

State variables written after the call(s)

```
284         _tokenId = 0;
```

The impact is limited since the `_tokenId` is fixed during the function call, and the function `onTokenBurned` requires `msg.sender == address(xenCrypto)`.

Even though the `tokenId` can be changed in the function `bulkClaimRank`, this function has no restrictions thus the impact is limited.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[XEN Team]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/FairCrypto/XENFT/commit/419013286117de1fd3d65b364fe8c2b81ca37b73

uint256 private _tokenId; this var is used as a reentrancy guard for functions where _safeMint is used; uint256 private constant _NOT_USED = 2**256 - 1; // 0xFF..FF uint256 private constant _USED = _NOT_USED - 1; // 0xFF..FE these constants above denote corresponding state.

the same _tokenId var is also used as state-keeping/reentrancy guard when transactional call to burn() function is finalized in a callback function onTokenBurned(), to check state correctness and also to prevent reentrancy.

Extensive tests for reentrancy guards have been also added to the repo

We have also followed your suggestion on adhering to Checks-Effects-Interactions pattern. Re _tokenId being changed after the call (Interaction), it's justified since it's been used as reentrancy/state guard

GLOBAL-02 | QUESTION RELATED TO TOKENOMICS

Category	Severity	Location	Status
Logical Issue	Informational		Acknowledged

Description

There are no limits on user-owned NFTs and VMUs. A user can create a large number of VMUs to claim more rare tokens.

Recommendation

We understand it might be an intended potential design, we still recommend the team to provide illustrations on this.

Alleviation

Issue acknowledged. I won't make any changes for the current version.

Indeed, the design is intended. XENFTs serve as a utility NFTs, acting as access tokens to minting pools of VMUs. Therefore there's not point in limiting their quantity. However special categories (Limited and Apex) bear rarity properties based on the limited supply (only 10,000 of Apex is possible) or limited time (Limited XENFTs are only available for 1 year since genesis). Number of VMUs is not directly influencing the amount of possible minted XENFTs; rather its combination with time (for Collector category) or amount of burned XEN (for Limited and Apex categories). Also, max number of VMUs (per different chain) is limited by the chain block gas limit; for example limit of VMUs for Polygon is ~100, whereas for Ethereum mainnet it's ~128

XEF-01 | DIFFERENT SOLIDITY VERSIONS

Category	Severity	Location	Status
Language Specific	Informational	projects/XENFT-master/contracts/XENFT.sol: 2; projects/XENFT-master/contracts/interfaces/IERC2771.sol: 2; projects/XENFT-master/contracts/interfaces/IXENProxying.sol: 2; projects/XENFT-master/contracts/interfaces/IXENTorrent.sol: 2; projects/XENFT-master/contracts/libs/Array.sol: 2; projects/XENFT-master/contracts/libs/BokkyPooBahsDateTimeLibrary.sol: 2; projects/XENFT-master/contracts/libs/DateTime.sol: 2; projects/XENFT-master/contracts/libs/ERC2771Context.sol: 4; projects/XENFT-master/contracts/libs/FormattedStrings.sol: 2; projects/XENFT-master/contracts/libs/Metadata.sol: 2; projects/XENFT-master/contracts/libs/MintInfo.sol: 2; projects/XENFT-master/contracts/libs/SVG.sol: 2; projects/XENFT-master/contracts/libs/StringData.sol: 2	Resolved

Description

Multiple Solidity versions are used in the codebase.

Versions used: `^0.8.1`, `^0.8.13`, `^0.8.10`, `>=0.6.0<0.9.0`, `^0.8.9`, `^0.8.0`

`^0.8.0` is used in node_modules/abdk-libraries-solidity/ABDKMath64x64.sol file.

```
6  pragma solidity ^0.8.0;
```

`^0.8.1` is used in node_modules/@openzeppelin/contracts/utils/Address.sol file.

```
4  pragma solidity ^0.8.1;
```

`^0.8.13` is used in node_modules/operator-filter-registry/src/OperatorFilterer.sol file.

```
2  pragma solidity ^0.8.13;
```

`^0.8.10` is used in node_modules/@faircrypto/xen-crypto/contracts/interfaces/IStakingToken.sol file.

```
2  pragma solidity ^0.8.10;
```

`>=0.6.0<0.9.0` is used in projects/XENFT-master/contracts/libs/BokkyPooBahsDateTimeLibrary.sol file.

```
2  pragma solidity >=0.6.0 <0.9.0;
```

`^0.8.9` is used in projects/XENFT-master/contracts/libs/ERC2771Context.sol file.

```
4  pragma solidity ^0.8.9;
```

Recommendation

We recommend using one Solidity version.

Alleviation

[XEN Team]: Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/FairCrypto/XENFT/commit/419013286117de1fd3d65b364fe8c2b81ca37b73

we decided not to change Solidity version in any of the imported libraries/contracts.

However, we did change version of Solidity to 0.8.10 to all contracts, libraries and interfaces that are part of the current repo

XEF-02 | INCONSISTENCY BETWEEN CODE AND DOCUMENTATION

Category	Severity	Location	Status
Logical Issue	Informational	projects/XENFT-master/contracts/XENFT.sol: 73~85; projects/XENFT-master/contracts/libs/StringData.sol: 46~64	Resolved

Description

We noticed there is a mismatch between the code and the whitepaper of XEN crypto (https://faircrypto.org/xenft_litepaper.pdf). The whitepaper states that `Limited` XENFTs requires burning XEN tokens to mint, while the comment states that `specialSeriesBurnRates[Limited]` is 0.

Recommendation

We recommend to revisit the design and make sure code is working as intended, or update the public documentation.

Alleviation

[XEN Team]: Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/FairCrypto/XENFT/commit/419013286117de1fd3d65b364fe8c2b81ca37b73

Litepaper is correct. Comment in the contract code L82 has been added. Just to confirm: specialSeriesBurnRates[Limited] > 0

OPTIMIZATIONS | XEN CRYPTO - AUDIT 2

ID	Title	Category	Severity	Status
BPB-02	Unused State Variable	Gas Optimization	Optimization	Resolved

BPB-02 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	projects/XENFT-master/contracts/libs/BokkyPooBahsDateTimeLibrary.sol: 35, 36, 37, 38, 41	Resolved

Description

One or more state variables are never used in the codebase.

Variable `_DOW_MON` in `BokkyPooBahsDateTimeLibrary` is never used in `BokkyPooBahsDateTimeLibrary`.

```
35      uint256 constant _DOW_MON = 1;
```

```
29  library BokkyPooBahsDateTimeLibrary {
```

Variable `_DOW_TUE` in `BokkyPooBahsDateTimeLibrary` is never used in `BokkyPooBahsDateTimeLibrary`.

```
36      uint256 constant _DOW_TUE = 2;
```

```
29  library BokkyPooBahsDateTimeLibrary {
```

Variable `_DOW_WED` in `BokkyPooBahsDateTimeLibrary` is never used in `BokkyPooBahsDateTimeLibrary`.

```
37      uint256 constant _DOW_WED = 3;
```

```
29  library BokkyPooBahsDateTimeLibrary {
```

Variable `_DOW_THU` in `BokkyPooBahsDateTimeLibrary` is never used in `BokkyPooBahsDateTimeLibrary`.

```
38      uint256 constant _DOW_THU = 4;
```

```
29  library BokkyPooBahsDateTimeLibrary {
```

Variable `_DOW_SUN` in `BokkyPooBahsDateTimeLibrary` is never used in `BokkyPooBahsDateTimeLibrary`.

```
41      uint256 constant _DOW_SUN = 7;
```

```
29  library BokkyPooBahsDateTimeLibrary {
```

Recommendation

We advise removing the unused variables.

Alleviation

[XEN Team]: the mentioned variables were not state variables, but constants. they don't consume gas, and they were part of the original code of a 3rd party library

However we decided to take your proposal and remove them for extra code cleanness.

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/FairCrypto/XENFT/commit/419013286117de1fd3d65b364fe8c2b81ca37b73

APPENDIX | XEN CRYPTO - AUDIT 2

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.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

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.