CERTIK

Security Assessment

The Space

May 26th, 2022

Table of Contents

Summary

This report has been prepared for The Space to discover issues and vulnerabilities in the source code of the The Space 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:

- 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.

Overview

Project Summary

Project Name	The Space
Platform	EVM Compatible
Language	Solidity
Codebase	https://github.com/thematters/contracts/
Commit	369035326f4a217490c719fe99583d92631c98e0 66d1cff1489e52946d215749bec4a8048ceda84a

Audit Summary

Delivery Date	May 26, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

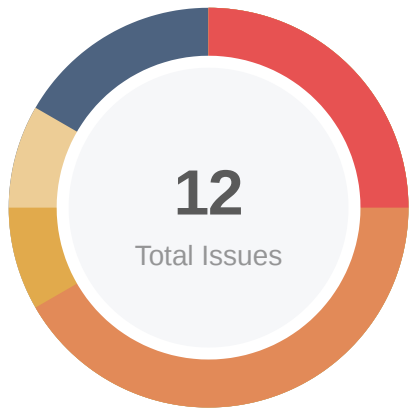
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	3	0	0	0	0	0	3
Major	5	0	0	1	3	0	1
Medium	1	0	0	0	0	0	1
Minor	1	0	0	0	0	0	1
Informational	2	0	0	1	0	0	1
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IAC	IACLManager.sol	0f0d2e53d548f6a50de851d54761fe1ddaa820ef9807386567e67045c326b008
STT	SpaceToken.sol	608ecadb98819010caf86a553e219149b92561f31c08226137a2b3ed6f0bbf58
TSR	TheSpaceRegistry.sol	313c721f7c58e60b4749b802cf007f9e43cb21644d2fc920e8ce7634559ff805
ACL	ACLManager.sol	844a1a2dbc0c5e53173cf19cb524c0f119b667e5a42fbabdd10f8682366db0c6
ITR	ITheSpaceRegistry.sol	f1540661adcbc4c1c8669b5b571b20845d07291c1d8938b715e7fd448d4d05ad
TST	TheSpace.sol	cc38419214a3c4e228b71580e681eaecd5ac91b941f93187558562a0597594a6
ITS	ITheSpace.sol	6889f6d14ef3250b1f05a94c56382e061a006a98922493a698ab5bc05ec0fef8

Findings



Critical	**3** (25.00%)
Major	**5** (41.67%)
Medium	**1** (8.33%)
Minor	**1** (8.33%)
Informational	**2** (16.67%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
ACL-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Mitigated
STT-01	Initial Token Distribution	**Centralization / Privilege**	**Major**	Mitigated
TSR-01	Missing `_isApprovedOrOwner` Check In Public Function `safeTransferFrom`	Logical Issue	Critical	Resolved
TSR-02	ERC-721 Reentrancy Attack In Function `_safeTransfer()`	Logical Issue	Major	Resolved
TSR-03	No Upper Limit For Tax	Control Flow	Major	Acknowledged
TSR-04	ERC-721 Reentrancy Attack In Function `_safeMint()`	Logical Issue	Medium	Resolved
TSR-05	Missing Emit Events	Coding Style	Informational	Resolved
TST-01	Missing Access Restriction	Logical Issue	Critical	Resolved
TST-02	Reentrancy Issue In Function `Bid()` Makes ERC-721 Tokens *Flashloanable*	Logical Issue	Critical	Resolved
TST-03	Centralized Control Of Ownership Upgrade	**Centralization / Privilege**	**Major**	Mitigated
TST-04	Unchecked Value Of ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
TST-05	ERC-721 Holders Have To Keep Enough Currency Balance To Prevent Their Tokens Being Burnt Through Public Function `settleTax()`	Logical Issue	Informational	Acknowledged

ACL-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	ACLManager.sol: 42~47	Mitigated

Description

In the contract TheSpace.sol the role `marketAdmin` has authority over the following functions:

- function setTotalSupply()
- function setTaxConfig()

In the contract TheSpace.sol the role `treasuryAdmin` has authority over the following functions:

- function withdrawTreasury()

Any compromise to the privileged accounts may allow a hacker to take advantage of this authority and modify key settings of the registry contract.

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., multi-signature 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.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[The Space Team]:

We are organizing a DAO to manage and own The Space.

After launching, we will transfer the two admin roles to time lock contract, and assign multi sign addresses as the owner of time lock contract.

We will upload the evidence once the roles are transferred.

Multi-signature

Platform: POLYGON

Multi-sign proxy address:

https://polygonscan.com/address/0x897b6dcbdf36045fe9c756288d1ec69cb43728b6

Transaction proof for transferring ownership to multi-signature proxy:

https://polygonscan.com/tx/0x6a14f8fa15728372b47720e3475a043342a0e0feca29626342fa79f7e3644ecd

Internal multi-signature address:
https://polygonscan.com/address/0x73d5f95B8f6FC4178aA934c44D7Acc7dC2a6DAf8

https://polygonscan.com/address/0x6d387216e10f346F02440FD26AD7BbC9Df0b54C5

https://polygonscan.com/address/0x869871ba7eC8BCB59530a04e1e28A42A208a697b

STT-01 | Initial Token Distribution

Category	Severity	Location	Status
Centralization / Privilege	**Major**	SpaceToken.sol: 11~13	Mitigated

Description

All of the **SPACE** tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute **SPACE** tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Alleviation

[The Space Team]:

We are now minting SPACE token according to the distribution of token specified in white paper.

In this commit we are minting token to treasury address and team address. As our white paper evolves, we will add in the other allocations.

See: whitepaper

TSR-01 | Missing `_isApprovedOrOwner` Check In Public Function `safeTransferFrom`

Category	Severity	Location	Status
Logical Issue	Critical	TheSpaceRegistry.sol: 206~230	Resolved

Description

In the current implementation of the public function `safeTransferFrom()`, there is no validation check if the caller is the owner of the function argument `token_Id` or has been approved.

```
217    function safeTransferFrom(
218          address from_,
219          address to_,
220          uint256 tokenId_,
221          bytes memory data_
222      ) public override(ERC721, IERC721) {
223          ITheSpace market = ITheSpace(owner());
224
225          bool success = market.beforeTransferByRegistry(tokenId_);
226
227          if (success) {
228              _safeTransfer(from_, to_, tokenId_, data_);
229          }
230      }
```

This will allow function caller to transfer arbitrary ERC-721 tokens owned by any accounts to any addresses, which will lead to devastating consequences if exploited by a hacker.

Recommendation

We recommend adding `_isApprovedOrOwner()` checks in public function `safeTransferFrom()` to mitigate this risk.

Alleviation

The team heeded our advice and fix the issue in this commit.

TSR-02 | ERC-721 Reentrancy Attack In Function `_safeTransfer()`

Category	Severity	Location	Status
Logical Issue	Major	TheSpaceRegistry.sol: 184~190, 217~230	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.

The `_safeTransfer()` is inherited from openzeppelin's ERC721 contract, which implements a `_checkOnERC721Received()` call back function to the caller contract.

```
1      function _safeTransfer(
2          address from,
3          address to,
4          uint256 tokenId,
5          bytes memory data
6      ) internal virtual {
7          _transfer(from, to, tokenId);
8          require(_checkOnERC721Received(from, to, tokenId, data), "ERC721: transfer to
non ERC721Receiver implementer");
9      }
10
```

This issue will lead to serious consequences, as we mentioned in finding **TST-02**.

Recommendation

One possible way to mitigate the issue is to disallow smart contracts to receive the NFTs if the team doesn't intend to do so. In this case, the following "_isContract" check can be used on function argument `to_`.

```
function _isContract(address addr) internal view returns (bool) {
    uint256 size;
    assembly {
    size := extcodesize(addr)
    }
    return size > 0;
}
```

Alleviation

This issue was mitigated as the team fixed TST-02.

TSR-03 | No Upper Limit For Tax

Category	Severity	Location	Status
Control Flow	Major	TheSpaceRegistry.sol: 88~92	Acknowledged

Description

There are no upper limits restricting parameters of `setTaxconfig()`, potentially enabling up to 100% of tax.

Recommendation

We recommend setting an upper limit for every variable mentioned.

Alleviation

[The Space Team]:

The tax parameter is percentage of tax per block number. Therefore, after a sufficient block number, the tax does indeed accumulate up to 100%. We are not yet aware of theoretical or practical limits on tax rate for Harberger tax, so we will keep the judgement to the managing DAO.

TSR-04 | ERC-721 Reentrancy Attack In Function `_safeMint()`

Category	Severity	Location	Status
Logical Issue	Medium	TheSpaceRegistry.sol: 175	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.

The `_safeMint()` is inherited from openzeppelin's ERC721 contract, which implements a `_checkOnERC721Received()` call back function to the caller contract.

```
1     function _safeMint(
2         address to,
3         uint256 tokenId,
4         bytes memory _data
5     ) internal virtual {
6         _mint(to, tokenId);
7         require(
8             _checkOnERC721Received(address(0), to, tokenId, _data),
9             "ERC721: transfer to non ERC721Receiver implementer"
10        );
11    }
```

This issue is considered benign because the reentrancy only acts as a double call in the current implementation. But this issue may lead to a worse scenario if the team expanded the functionalities in later commits.

Recommendation

One possible way to mitigate the issue is to disallow smart contracts to receive the minted NFTs if the team doesn't intend to do so. In this case, the following "_isContract" check can be used on function argument `to_`.

```
function _isContract(address addr) internal view returns (bool) {
    uint256 size;
    assembly {
    size := extcodesize(addr)
```

```
    }
    return size > 0;
}
```

Alleviation

This issue was mitigated as the team fixed TST-02.

TSR-05 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	Informational	TheSpaceRegistry.sol: 83, 95, 104, 114, 173, 179, 184, 237, 242	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

The team heeded our advice and fix the issue in this commit.

TST-01 | Missing Access Restriction

Category	Severity	Location	Status
Logical Issue	Critical	TheSpace.sol: 113~120	Resolved

Description

In the current implementation of function `_setColor`, there is no `isApprovedOrOwner` check on the function caller, which means the caller can set the color of any ERC-721 tokens that are owned by others.

```
113    function _setColor(
114          uint256 tokenId_,
115          uint256 color_,
116          address owner_
117      ) public {
118          registry.setColor(tokenId_, color_);
119          registry.emitColor(tokenId_, color_, owner_);
120      }
```

Recommendation

We recommend setting the `_setColor` to **internal** to mitigate the aforementioned risks.

Alleviation

The team heeded our advice and fix the issue in this commit.

TST-02 | Reentrancy Issue In Function `Bid()` Makes ERC-721 Tokens *Flashloanable*

Category	Severity	Location	Status
Logical Issue	Critical	TheSpace.sol: 208~259	Resolved

Description

In the current implementation of function `bid()`, The ERC-20 token transfer in L252(Paying the price) is conducted after the `registry.safeTransferByMarket` in L237 and L245 (ERC-721 Token transfer).

```
208    function bid(uint256 tokenId_, uint256 price_) public {
209          address owner = getOwner(tokenId_);
210          uint256 askPrice = _getPrice(tokenId_);
211          uint256 mintTax =
registry.taxConfig(ITheSpaceRegistry.ConfigOptions.mintTax);
212
213          // bid price and payee is calculated based on tax and token status
214          uint256 bidPrice;
215          address payee;
216
217          if (registry.exists(tokenId_)) {
218              // skip if already own
219              if (owner == msg.sender) return;
220
221              // clear tax
222              bool success = _collectTax(tokenId_);
223
224              // process with transfer
225              if (success) {
226                  // if tax fully paid, owner get paid normally
227                  bidPrice = askPrice;
228                  payee = owner;
229              } else {
230                  // if tax not fully paid, token is treated as defaulted and mint tax
is collected and recorded
231                  bidPrice = mintTax;
232                  payee = address(registry);
233                  _recordTax(tokenId_, msg.sender, mintTax);
234              }
235
236              // settle ERC721 token
237              registry.safeTransferByMarket(owner, msg.sender, tokenId_);
238          } else {
239              // mint tax is collected and recorded
240              bidPrice = mintTax;
```

```
241             payee = address(registry);
242             _recordTax(tokenId_, msg.sender, mintTax);
243
244             // settle ERC721 token
245             registry.mint(msg.sender, tokenId_);
246         }
247
248         // revert if price too low
249         if (price_ < bidPrice) revert PriceTooLow();
250
251         // settle ERC20 token
252         registry.transferCurrencyFrom(msg.sender, payee, bidPrice);
253         // emit bid event
254         registry.emitBid(tokenId_, owner, msg.sender, bidPrice);
255
256         // update price to ask price if difference
257         if (price_ > askPrice) _setPrice(tokenId_, price_);
258     }
```

Due to the reentrancy risks in ERC-721 `_safeMint()` and `_safeTransfer` as we mentioned in other findings, the current design actually allows anyone to `Flashloan` ERC-721 tokens.

Here is what the hacker can do:

- The attacker's contract calls bid() to temporarily gain the ownership of ERC-721 tokens.
- In the `_OnERC721Received` _OnERC721Received in attacker's contract, the attacker performs another call of `Bid` to gain more control of the ERC-721 tokens. He can repeat this process until he gains enough tokens. (He does not have to pay the price for these tokens because he just repeats the execution between L208-237)
- In the `_OnERC721Received` function, the attacker can implement arbitrary code to use the `flashloaned` tokens to perform flash loan attacks, for example, claiming UBI through function `withdrawUbi()`, attacking other contracts in this project, manipulating the token prices in price oracle, or doing the arbitrage.
- Paying the token price using his earns from the last step.

Allowing ERC-721 tokens to be "flashloanable" is dangerous and this will lead to serious consequences.

Recommendation

We recommend rearranging the ERC-20 token transfer and related states updating (L249-L257) before the ERC-721 token transfer in L237 and L245.

Besides, We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

The team heeded our advice and fix the issue in this commit.

TST-03 | Centralized Control Of Ownership Upgrade

Category	Severity	Location	Status
Centralization / Privilege	**Major**	TheSpace.sol: 43~45	Mitigated

Description

Function `upgradeTo()` allows the role `aclManager` to transfer the ownership of contract `TheSpaceRegistry` to an arbitrary address.

```
43    function upgradeTo(address newImplementation) external onlyRole(Role.aclManager) {
44          registry.transferOwnership(newImplementation);
45      }
```

The role `aclManager` can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the Ownership of the ERC-721 contract and drain tokens from the contract.

Recommendation

We advise the client to carefully manage the admin account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multi-signature wallets. Here are some feasible solutions that would also mitigate the potential risk:

- **Time-lock with reasonable latency, i.e. 24*7 hours, for awareness on upgrade operations;**
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[The Space Team]:

We are in the process of setting up a DAO governance structure. After launching, the upgrade operation role will be transferred to a time-lock contract that is controlled by a multi-signature wallet.

The multi-sig contract is deployed here:
https://polygonscan.com/address/0x897B6DcBdf36045FE9C756288D1Ec69CB43728b6

TST-04 | Unchecked Value Of ERC-20 `transfer()` / `transferFrom()` Call

Category	Severity	Location	Status
Volatile Code	Minor	TheSpace.sol: 252	Resolved

Description

The linked `transfer()`/`transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

The aforementioned lines perform external calls to `transferFrom` of ERC20 contracts and the return value is not checked in either case.

Recommendation

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation

[The Space Team]:

The ERC20 token we are using are inherited directly from OpenZeppelin's ERC20.sol, and the token address is immutable. Therefore, we believe this risk does not apply in this case.

TST-05 | ERC-721 Holders Have To Keep Enough Currency Balance To Prevent Their Tokens Being Burnt Through Public Function `settleTax()`

Category	Severity	Location	Status
Logical Issue	Informational	TheSpace.sol: 351~354	Acknowledged

Description

In the current implementation, ERC-721 token holders must have to keep enough currency balance in their account to pay for the tax, or their tokens will be burnt through the public function `settleTax`.

```
192    function settleTax(uint256 tokenId_) public returns (bool success) {
193          success = _collectTax(tokenId_);
194          if (!success) registry.burn(tokenId_);
195      }
196
```

Recommendation

If the current implementation is the intended design, we advise the team to present the facts and potential risks to the community in a clear and notable fashion.

Alleviation

[The Space Team]:

This is one major rule of Harberger Tax economic model, and the risk is also made clear in our frontend UI/UX design. Users try to maximize their profit while minimizing the risk of getting defaulted.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

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.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

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.

Disclaimer

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.

About

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.