CERTIK

Security Assessment

SaitaMask

Dec 25th, 2021

Table of Contents

Summary

This report has been prepared for SaitaMask to discover issues and vulnerabilities in the source code of the SaitaMask 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	SaitaMask
Platform	Ethereum
Language	Solidity
Codebase	git@gitlab.com:luismauricio/saitamask-v1-core.git git@gitlab.com:luismauricio/saitamask-v1-periphery.git
Commit	21ea25ea678827664af0caa8a3e04e710e067e52 82604f2f2414b9deb98a29ca6d32c86b29904244

Audit Summary

Delivery Date	Dec 25, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

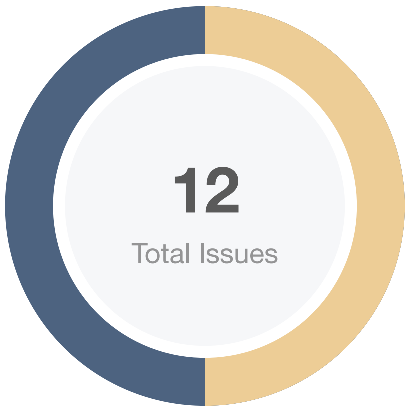
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● Major	0	0	0	0	0	0
● Medium	0	0	0	0	0	0
● Minor	6	0	0	2	0	4
● Informational	6	0	0	0	0	6
● Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
MSM	projects/SaitaMask/saitamask-v1-core/contracts/Migrations.sol	4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4
SVE	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1ERC20.sol	6c3aa1127016c57f5a9292129636b16f33f805fcd6b90c1e11352fca3d21b2f0
SVF	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1Factory.sol	4dd15c2fbc8a9e837e5b36920a21219de20f08d79f126341cfc0c622f6870723
SVP	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1Pair.sol	d5a4ae472a3d07ae8b606cf8cd830400abf50d63d201472e0919d3e0a97abd24
BSM	projects/SaitaMask/saitamask-v1-periphery/contracts/libraries/Babylonian.sol	55f7f97f332b408835ff07a374bef0a8ef698abe282de76a259f2ea6b7d210b4
SVL	projects/SaitaMask/saitamask-v1-periphery/contracts/libraries/SaitamaskV1Library.sol	540b508bf172c6b78776d91fe70f4e9ac17c816c0d2ca07ae4c993555880633b
SVM	projects/SaitaMask/saitamask-v1-periphery/contracts/libraries/SaitamaskV1LiquidityMathLibrary.sol	8735ef8f74fd58a490f40cd5660ffb403fb997357f28765c9f7362fabccd0719
SVO	projects/SaitaMask/saitamask-v1-periphery/contracts/libraries/SaitamaskV1OracleLibrary.sol	e634ed159a1bf2ad9b8af0b6dcec3ec56bc2f28b16515a0ffa4ffde8fed78464
THS	projects/SaitaMask/saitamask-v1-periphery/contracts/libraries/TransferHelper.sol	22b87fd425d590e533ab7e52478cf72bdc4bde2672e0977c7eff7742e8f0737d
SVS	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Migrator.sol	3fbbccd15b4fcf6324e61719a630fe06a048456dbc982879e0ac301598e79e76
SVR	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router01.sol	e89975e4a015bd2782977bc21ffbabe91cfcc59e7a4b6a6c503394cb28f5b402
SVC	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router02.sol	9cb30569401c20226d7096135b2036f4487e101cd5de6d4df2b4a1d646acbb75
WET	projects/SaitaMask/saitamask-v1-periphery/contracts/WETH.sol	a847d003c6497f43e244f4c2e2690a34313e96217e69db097ae839c573db1b7f

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**0** (0.00%)
🟨	**Medium**	**0** (0.00%)
🟨	**Minor**	**6** (50.00%)
🟦	**Informational**	**6** (50.00%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
SVE-01	Missing Input Validation	Volatile Code	● Minor	✓ Resolved
SVE-02	Declaration Naming Convention	Coding Style	● Informational	✓ Resolved
SVF-01	Unnecessary Array as Counter	Gas Optimization	● Informational	✓ Resolved
SVF-02	Missing Emit Events	Coding Style	● Informational	✓ Resolved
SVP-01	Divide by Zero	Logical Issue	● Minor	✓ Resolved
SVR-01	Missing Input Validation	Volatile Code	● Minor	✓ Resolved
SVR-02	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	✓ Resolved
SVR-03	Incompatibility With Deflationary Tokens	Logical Issue	● Minor	ⓘ Acknowledged
SVS-01	Missing Input Validation	Volatile Code	● Minor	✓ Resolved
SVS-02	Proper Usage of `require` And `assert` Functions	Coding Style	● Informational	✓ Resolved
SVS-03	Incompatibility With Deflationary Tokens	Logical Issue	● Minor	ⓘ Acknowledged
WET-01	Declaration Naming Convention	Coding Style	● Informational	✓ Resolved

SVE-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1ERC20.sol: 46, 52, 57	✓ Resolved

Description

The given input is missing the check for the non-zero address.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:

```
46  function _burn(address from, uint value) internal {
47      require(from != address(0), "burn from zero address!");
48      balanceOf[from] = balanceOf[from].sub(value);
49      totalSupply = totalSupply.sub(value);
50      emit Transfer(from, address(0), value);
51  }
```

```
52  function _approve(address owner, address spender, uint value) private {
53      require(owner != address(0), "owner is zero address!");
54      require(spender != address(0), "spender is zero address!");
55      allowance[owner][spender] = value;
56      emit Approval(owner, spender, value);
57  }
```

```
57  function _transfer(address from, address to, uint value) private {
58      require(from != address(0), "from is zero address!");
59      require(to != address(0), "to is zero address!");
60      balanceOf[from] = balanceOf[from].sub(value);
61      balanceOf[to] = balanceOf[to].add(value);
62      emit Transfer(from, to, value);
63  }
```

Alleviation

As per commit `82604f2f2414b9deb98a29ca6d32c86b29904244`, the affected functions are now checking for the non-zero address.

SVE-02 | Declaration Naming Convention

Category	Severity	Location	Status
Coding Style	● Informational	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1ERC20.sol: 9~11	✓ Resolved

Description

The linked declarations do not conform to the Solidity style guide with regards to its naming convention.

Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

Recommendation

We advise that the linked variable and function names are adjusted to properly conform to Solidity's naming convention.

Alleviation

As per commits `ba62ee0a2d481c171332c1664f52c82c370627ed` and `82604f2f2414b9deb98a29ca6d32c86b29904244`, the highlighted constants are now following the naming convention.

SVF-01 | Unnecessary Array as Counter

Category	Severity	Location	Status
Gas Optimization	● Informational	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1Factory.sol: 12, 20, 37, 38	✓ Resolved

Description

The usage of `allPairs` array is as a counter to maintain the number of created pairs.

Recommendation

We advise the client to replace the `allPairs` with a simple uint type counter to store the number of pairs created.

Alleviation

As per commit `82604f2f2414b9deb98a29ca6d32c86b29904244`, the `allPairs` array has been substituted by the `pairsCount` unsigned integer.

SVF-02 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	● Informational	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1Factory.sol: 41, 46	✓ Resolved

Description

The function that affects the status of sensitive variables should be able to emit events as notifications.

- `setFeeTo()`
- `setFeeToSetter()`

Recommendation

We advise the client to consider adding events for sensitive actions, and emit them in the function.

Alleviation

As per commit `82604f2f2414b9deb98a29ca6d32c86b29904244`, the `setFeeTo()` and `setFeeToSetter()` functions are now emitting events.

SVP-01 | Divide by Zero

Category	Severity	Location	Status
Logical Issue	● Minor	projects/SaitaMask/saitamask-v1-core/contracts/SaitamaskV1Pair.sol: 143~145	✓ Resolved

Description

If the value of `totalSupply` is 0, the following two division operations will fail due to the divide by 0 error, which ultimately make the invocation to `burn()` function fail.

```
amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
```

Recommendation

We advise the client to add the following validation in the function `burn()`

```
function burn(address to) external lock returns (uint amount0, uint amount1) {
   require(totalSupply != 0, "The value of totalSupply must not be 0");
    ...
}
```

Alleviation

As per commit `82604f2f2414b9deb98a29ca6d32c86b29904244`, the burn function is now checking whether the total supply is zero or not.

SVR-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router01.sol: 21~22	✓ Resolved

Description

The given input is missing the check for the non-zero address.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:

```
constructor(address _factory, address _WETH) public {
     require(_factory != address(0),"_factory should not be address(0)");
     require(_WETH != address(0),"_WETH should not be address(0)");
    factory = _factory;
    WETH = _WETH;
}
```

Alleviation

As per commit `ba62ee0a2d481c171332c1664f52c82c370627ed`, the constructor function is now making sure that the `_factory` and `_WETH` addresses are not equal to the zero address.

SVR-02 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router01.sol: 214, 256, 93, 52, 26	✓ Resolved

Description

The `assert()` function should only be used to test for internal errors, and to check invariants. The `require()` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require()` function, along with a custom error message when the condition fails, instead of the `assert()` function.

Alleviation

As per commit `ba62ee0a2d481c171332c1664f52c82c370627ed`, the `SaitamaskV1Router02.sol` is now using `require()` instead of `assert()`.

SVR-03 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	● Minor	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router01.sol: 70~71, 91, 109	ⓘ Acknowledged

Description

When users add or remove LP tokens into the router, and the `mint` and `burn` operations are performed. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, the amount inconsistency will occur and the transaction may fail due to the validation checks.

Recommendation

We advise the client to regulate the set of LP tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

SVS-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router02.sol: 24~25	✓ Resolved

Description

The given input is missing the check for the non-zero address.

Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:

```
constructor(address _factory, address _WETH) public {
     require(_factory != address(0),"_factory should not be address(0)");
     require(_WETH != address(0),"_WETH should not be address(0)");
     factory = _factory;
     WETH = _WETH;
}
```

Alleviation

As per commit `ba62ee0a2d481c171332c1664f52c82c370627ed`, the constructor function is now making sure that the `_factory` and `_WETH` addresses are not equal to the zero address.

SVS-02 | Proper Usage of `require` And `assert` Functions

Category	Severity	Location	Status
Coding Style	● Informational	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router02.sol: 29	Resolved

Description

The `assert()` function should only be used to test for internal errors, and to check invariants. The `require()` function should be used to ensure valid conditions, such as inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Recommendation

We advise the client using the `require()` function, along with a custom error message when the condition fails, instead of the `assert()` function.

Alleviation

As per commit `ba62ee0a2d481c171332c1664f52c82c370627ed`, the `SaitamaskV1Router02.sol` is now using `require()` instead of `assert()`.

SVS-03 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	● Minor	projects/SaitaMask/saitamask-v1-periphery/contracts/SaitamaskV1Router02.sol: 73~74, 94, 113	ⓘ Acknowledged

Description

When users add or remove LP tokens into the router, and the `mint` and `burn` operations are performed. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, the amount inconsistency will occur and the transaction may fail due to the validation checks.

Recommendation

We advise the client to regulate the set of LP tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

WET-01 | Declaration Naming Convention

Category	Severity	Location	Status
Coding Style	● Informational	projects/SaitaMask/saitamask-v1-periphery/contracts/WETH.sol: 23~25	✓ Resolved

Description

The linked declarations do not conform to the Solidity style guide with regards to its naming convention.

Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

Recommendation

We advise that the linked variable and function names are adjusted to properly conform to Solidity's naming convention.

Alleviation

As per commits `ba62ee0a2d481c171332c1664f52c82c370627ed` and `82604f2f2414b9deb98a29ca6d32c86b29904244`, the highlighted constants are now following the naming convention.

Appendix

Finding Categories

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.

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.

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.

CERTIK