CERTIK

Security Assessment

HELIO

May 30th, 2022

Table of Contents

Summary

This report has been prepared for HELIO to discover issues and vulnerabilities in the source code of the HELIO 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	HELIO
Platform	BSC
Language	Solidity
Codebase	https://github.com/helio-money/helio-smart-contracts

Audit Summary

Delivery Date	May 30, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

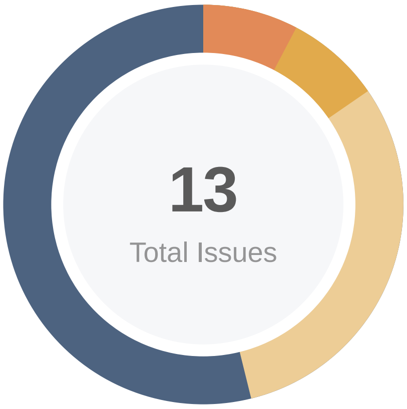
Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0	0
● Major	1	0	0	1	0	0	0
● Medium	1	0	0	1	0	0	0
● Minor	4	0	0	0	0	0	4
● Informational	7	0	0	0	0	0	7
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
DCM	contracts/DssCdpManager.sol	b4aa260b3eedc7579e1f54b512ee02e2b1afd5a8da0f2cbcdde3e2e3604aed4c
USB	contracts/usb.sol	ef95bd4805e605ace652ab4ff8f30d04c3affeb8303f2f0da252b6eb2f42477e
CLI	contracts/clip.sol	3bcbb65b4c572c57ef356dff11282ccf2521de565679109e525fca90e510b6d7
VAT	contracts/vat.sol	03d5b9f3d80b6935789a57d862877ebc40d25d088fb5fe4bbcdf4db801e3f9e0
CTB	contracts/ceros/CeToken.sol	3fc6f5874bc7c8d24a2d7c7fea4cffd5b9d562bfe85f7e1a5ff7e8d93c1f16c2
BNB	contracts/aBNBc.sol	ab3ec538fddb057fc053a90d369eb560d178ac38df7d576e1a3d33e1e5b33de4
FLA	contracts/flap.sol	243b74e041c282ee3caf34c3c586da306102d51b47d1616252474b623a5cd5a7
ICR	contracts/ceros/interfaces/ICerosRouter.sol	f4a8c344b627adef02b0021974cc4a0d3697eceaf994d01eb1ae472d04c3c71f
MBN	contracts/mBNB.sol	e4fa1ba831e90503e01e21e30b01875814b9696ea9b1648546dbfa241459156e
HMA	contracts/hMath.sol	c519e85f88f9f523e003d086f866aa3bff479e593d9d025e60ce0bb678b14759
CVB	contracts/ceros/CeVault.sol	15e6b83505c9a100e93e91c461e7f4ecf3aea19bd6df96a0c2d085a25c6559ed
QUE	contracts/Queue.sol	a28e5d5b587eb1afc0ea1f2264cb50494a5da4e43a1d57524ea680b3e7582556
IVB	contracts/ceros/interfaces/IVault.sol	bcdcc02f275826edb8d5e9f1a2f982d138051634997acae6f98b3f124833cf38
ICT	contracts/ceros/interfaces/ICertToken.sol	060dc9f5f8134caf8c926632494bf28b5404aa647c7bdb8c1c3afbf6f12f05a3
JUG	contracts/jug.sol	8cb595d586abf290623f9bb1c6b422dd9ec515d19c22eb45459866634c63d3a2
IHP	contracts/ceros/interfaces/IHelioProvider.sol	1315f352363d9a726cd439e2de95690162b063e39771605e626ebdd0b8b3a672
JAR	contracts/jar.sol	4d0b010c2b35f331ffb661f7e80db85d9db5eb7cc054992f9bc121f8fe25adfe

ID	File	SHA256 Checksum
JOI	contracts/join.sol	ae9d0993b1ba73f987253de858207079ddc5f48a00fe1af1f77fa141fea1f1bf
HPB	contracts/ceros/HelioProvider.sol	fcd284f040999ac23e8a9f6f04d540229aa705e805088c0c7286bbf24e35eca9
ABA	contracts/abaci.sol	3ab874780c8961fd9e3fe0c483ac1915915b17549257b4bd66411fb8e18585d7
SPO	contracts/spot.sol	4a19258c88909c7e8563133a17fe91451b17663194002bd498a2b7ed7ab9378b
POT	contracts/pot.sol	c43a02e587ae77d2e097fe4e91e244c75b28cd13f1160d807854c35eb83f622d
IDU	contracts/ceros/interfaces/IDex.sol	6bc2956e3fefa2f771b42bc3ce037469dec12c9fef0d1edc760b25f4f37d0390
CRB	contracts/ceros/CerosRouter.sol	129081f5b151d2e4a218dd6a38a16a083c1bad4655d75dd9a585b186d6eb5da1
BOB	contracts/oracle/BnbOracle.sol	dbec459a0c23bbfe91aaf8aa95cc30b44709ca748e1b917e9ac366eaa4d716a1
DOG	contracts/dog.sol	6ad8fdb5018a89fc2e3f370ab92af6331b2ff82d3af4940ee276c1ddaf7a7d97
HTB	contracts/HelioToken.sol	be741a64f3353a4c5e63e1e78c397c8fc18b689b25fc1b1e4bf99f0589ff578b
VOW	contracts/vow.sol	64df8d1ddb0704c5dbebceb9ee3b56429d42f05f257a8105cc14fda0dab1b7af
DAO	contracts/DAOInteraction.sol	0d2dabf1614d0afa4837d03d4f48949d1c41029165b03023457f4e11161420f2
IDB	contracts/ceros/interfaces/IDao.sol	80a06bf62bfad7d9fc65fa8ee789b2bea79e84d38602d646d6f3722a0965e284
FLO	contracts/flop.sol	ea1e1f195d64a5d9ad4ed832ab6934b20f78b4d8217179697c4edb386d2c953f
HRB	contracts/HelioRewards.sol	b9a431f209575e34402557b4322c2cdbce5617b93e75cd4cd88f3d047e4eaced

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**1** (7.69%)
🟨 **Medium**	**1** (7.69%)
🟨 **Minor**	**4** (30.77%)
🟦 **Informational**	**7** (53.85%)
🟩 **Discussion**	**0** (0.00%)

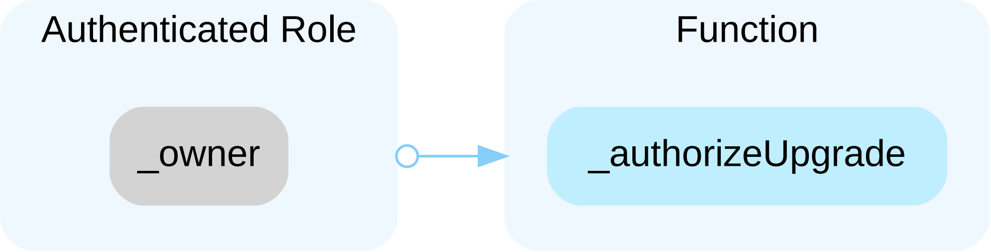
ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
CON-01	Potential Reentrancy Attack	Volatile Code	● Medium	ⓘ Acknowledged
CON-02	Duplication Of Code	Coding Style, Gas Optimization	● Informational	✓ Resolved
CON-03	Variables That Could Be Declared As Immutable	Gas Optimization	● Informational	✓ Resolved
CRB-01	Unchecked Variable Unit	Volatile Code	● Minor	✓ Resolved
CRB-02	Missing Approval Checks On `transferFrom()` Function	Volatile Code	● Minor	✓ Resolved
CRB-03	Inconsistent Comment And Code	Inconsistency	● Informational	✓ Resolved
DAO-01	Duplicated Validation	Gas Optimization	● Informational	✓ Resolved
HPB-01	Typo	Coding Style	● Informational	✓ Resolved
HRB-01	Duplication Of Code	Coding Style, Gas Optimization	● Minor	✓ Resolved
HRB-02	Usage Of Magic Number	Magic Numbers	● Informational	✓ Resolved
JAR-01	Lack Of Input Validation	Volatile Code	● Minor	✓ Resolved
USB-01	Missing Error Messages	Coding Style	● Informational	✓ Resolved

GLOBAL-01 | Centralization Related Risks

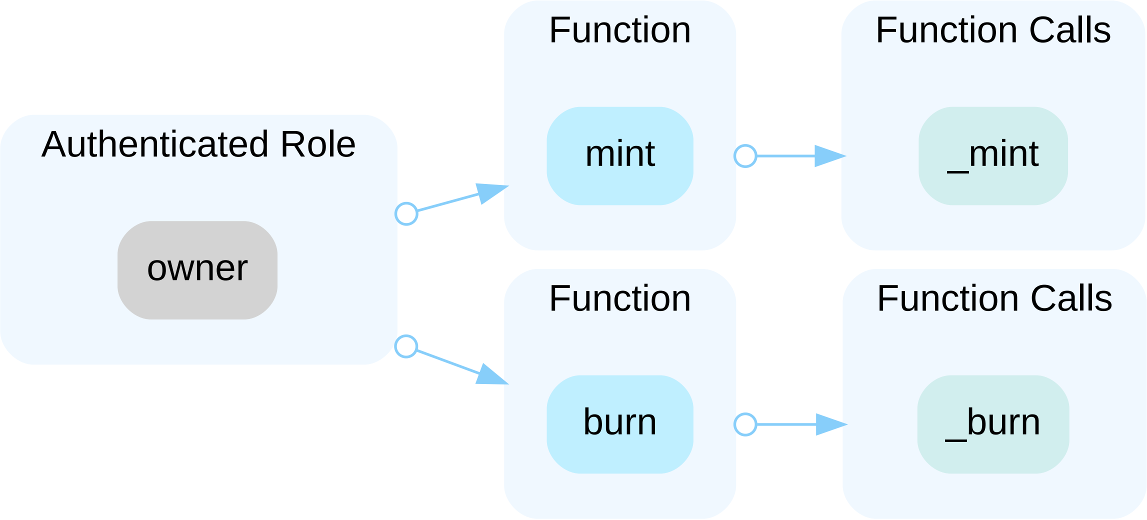
Category	Severity	Location	Status
Centralization / Privilege	● **Major**		ⓘ Acknowledged

Description

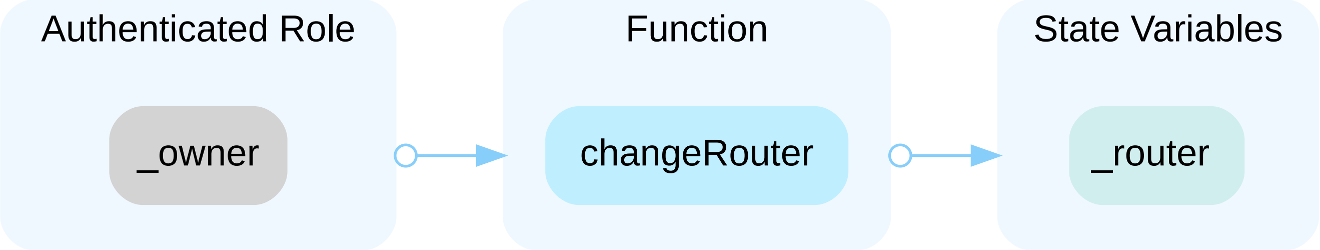
In the contract `DAOInteraction` the role `_owner` has authority over the functions shown in the diagram below.



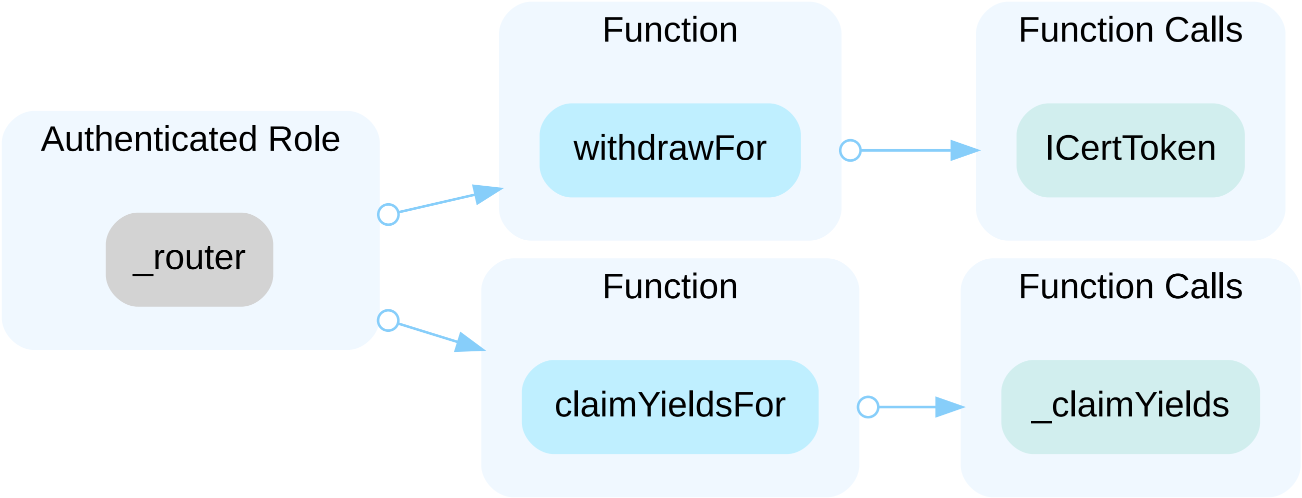
In the contract `aBNBc` the role `owner` has authority over the functions shown in the diagram below.



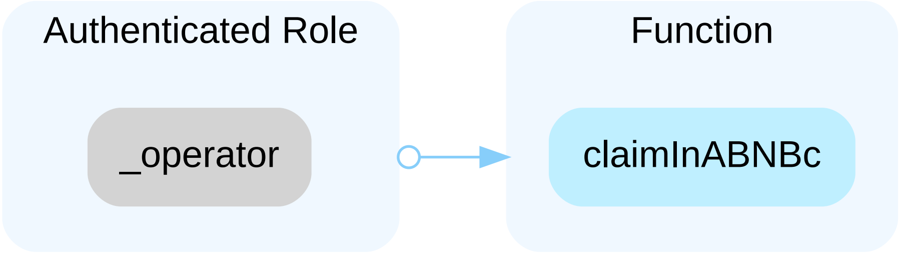
In the contract `CeVault` the role `_owner` has authority over the functions shown in the diagram below.



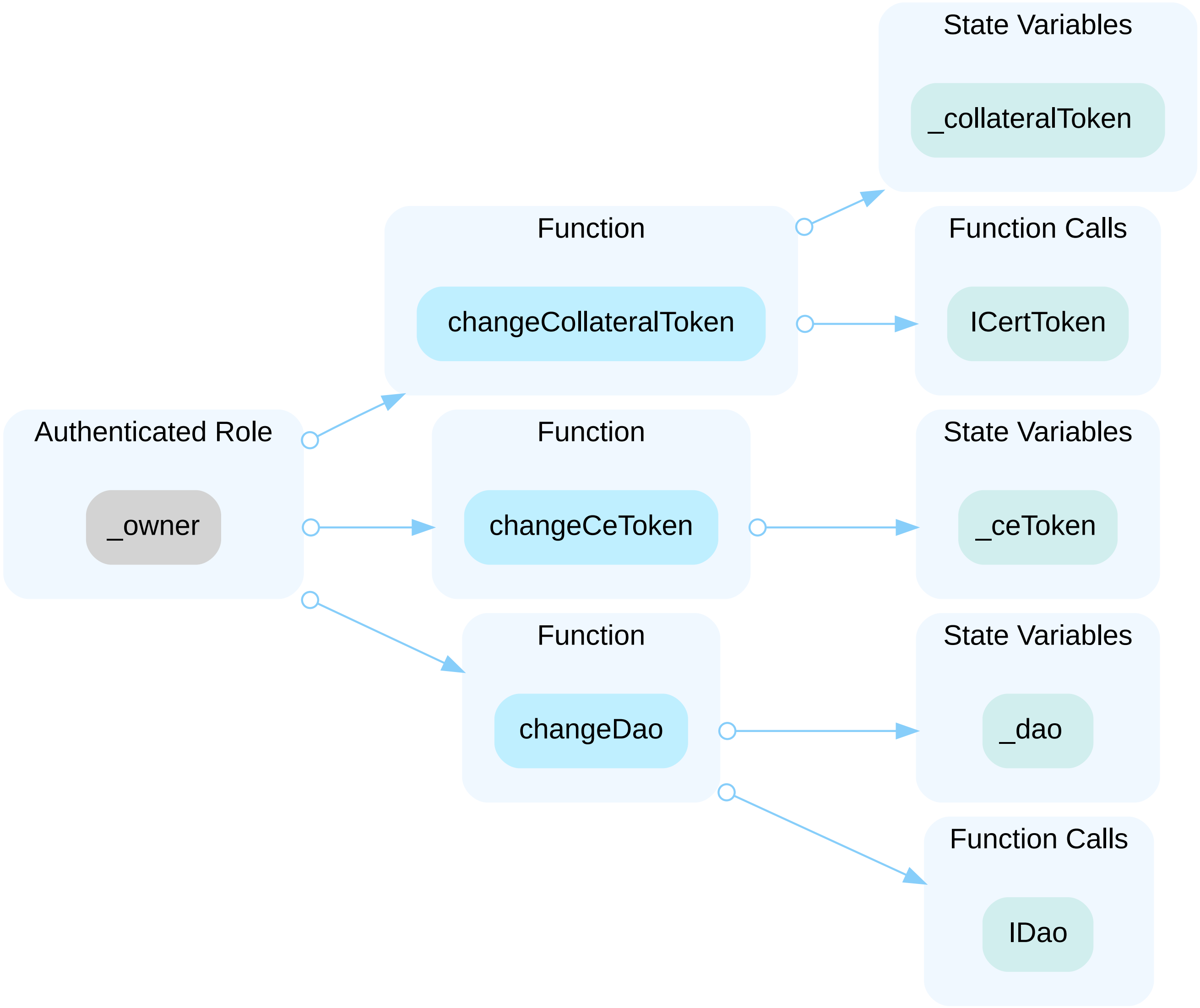
In the contract `CeVault` the role `_router` has authority over the functions shown in the diagram below.



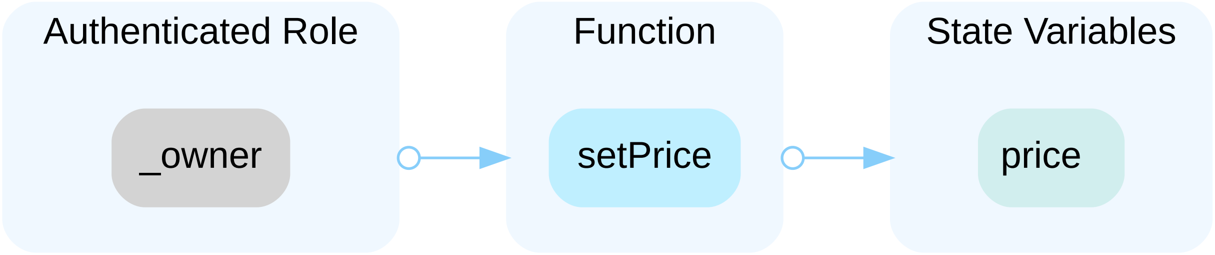
In the contract `HelioProvider` the role `_operator` has authority over the functions shown in the diagram below.



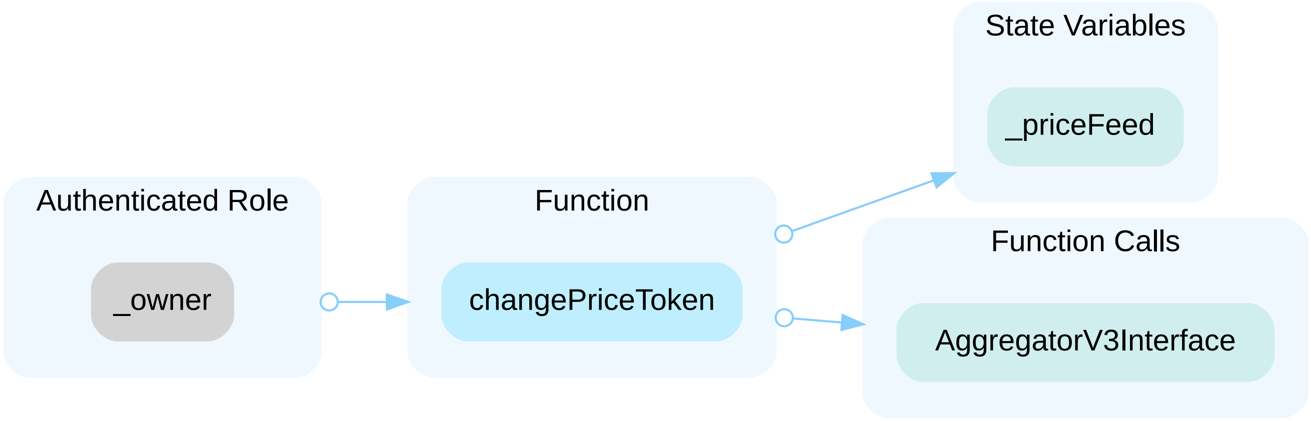
In the contract `HelioProvider` the role `_owner` has authority over the functions shown in the diagram below.



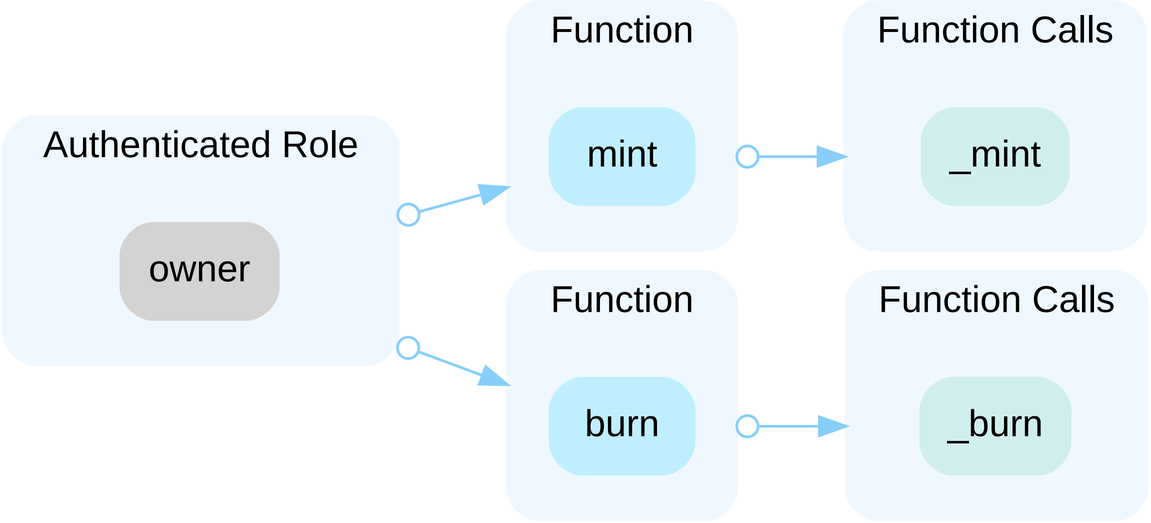
In the contract `Oracle` the role `_owner` has authority over the functions shown in the diagram below.



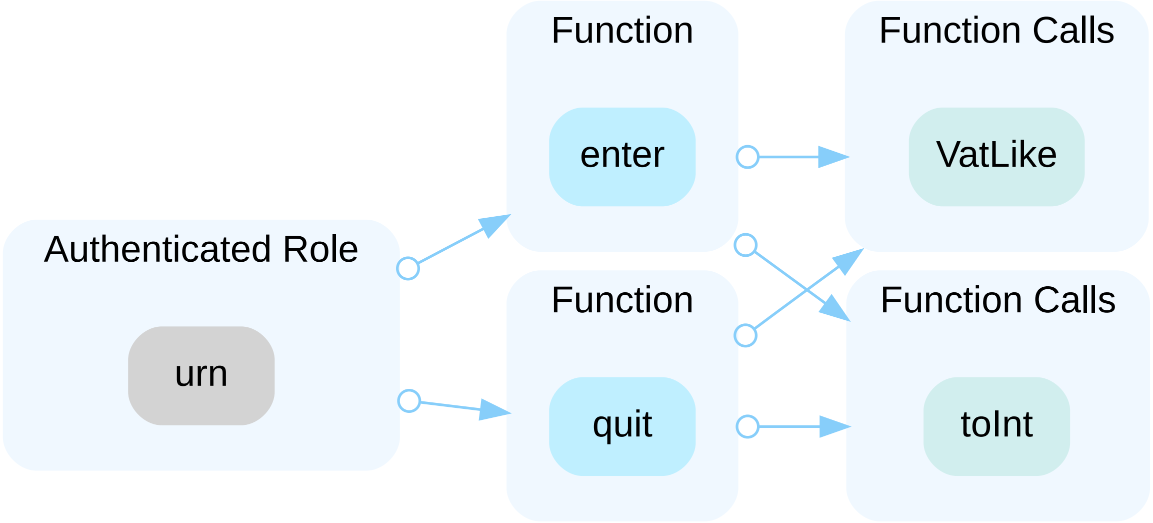
In the contract `BnbOracle` the role `_owner` has authority over the functions shown in the diagram below.



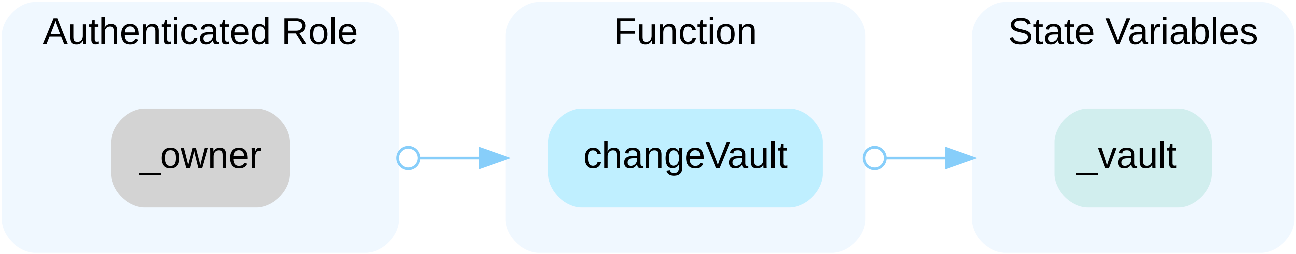
In the contract `aBNBc` the role `owner` has authority over the functions shown in the diagram below.



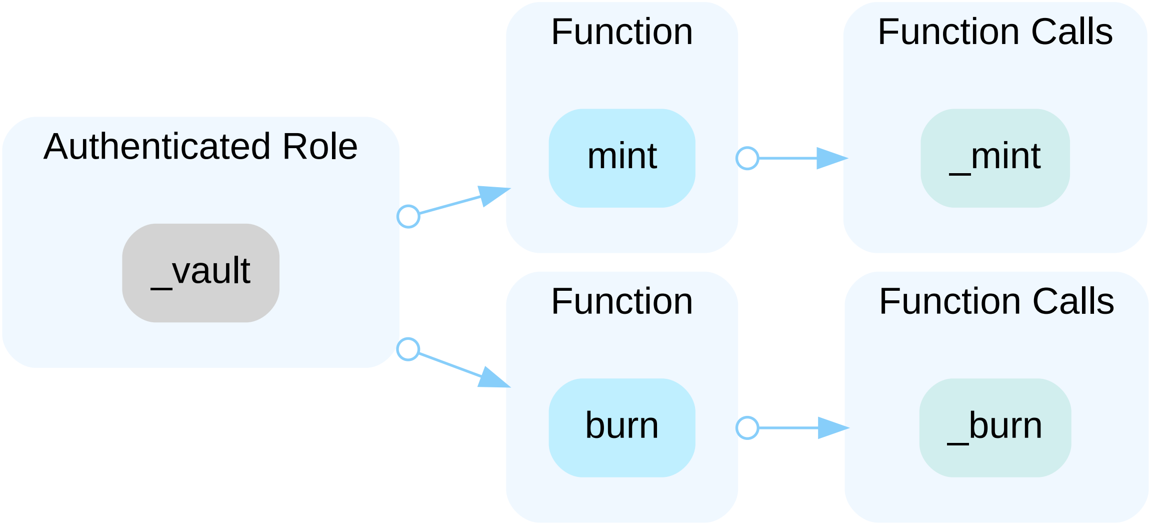
In the contract `DssCdpManager` the role `urn` has authority over the functions shown in the diagram below.



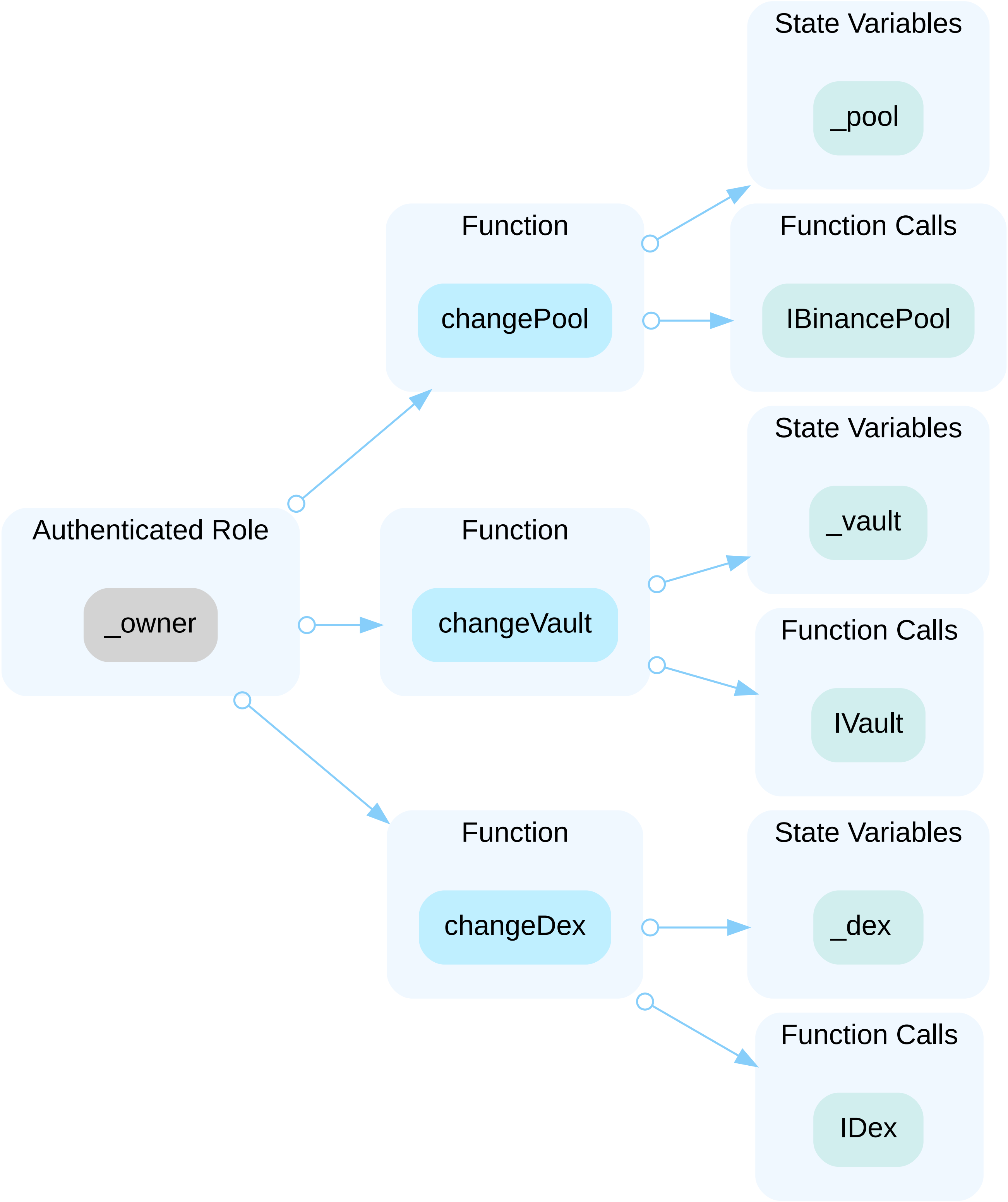
In the contract `CeToken` the role `_owner` has authority over the functions shown in the diagram below.



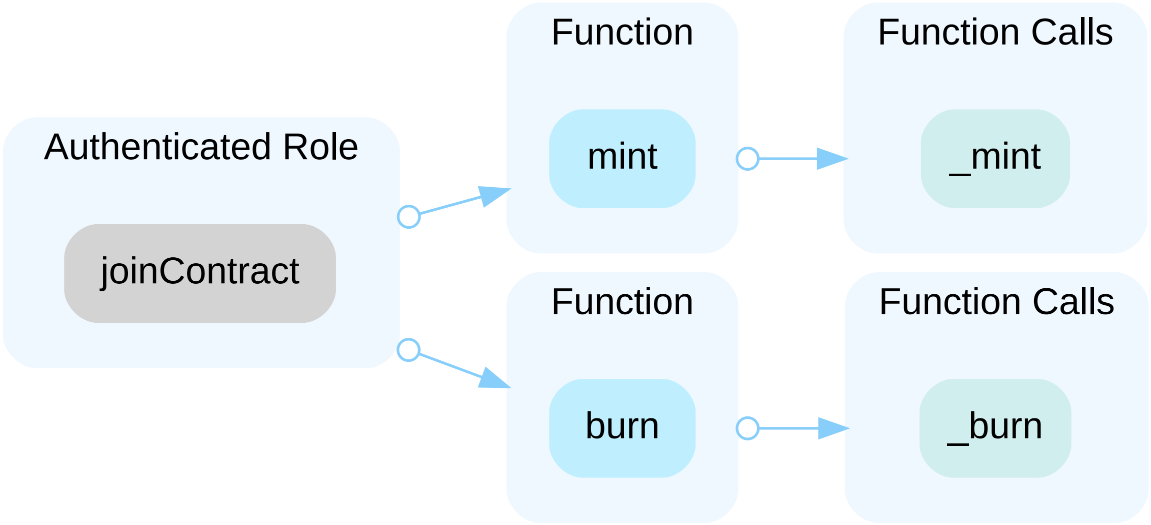
In the contract `CeToken` the role `_vault` has authority over the functions shown in the diagram below.



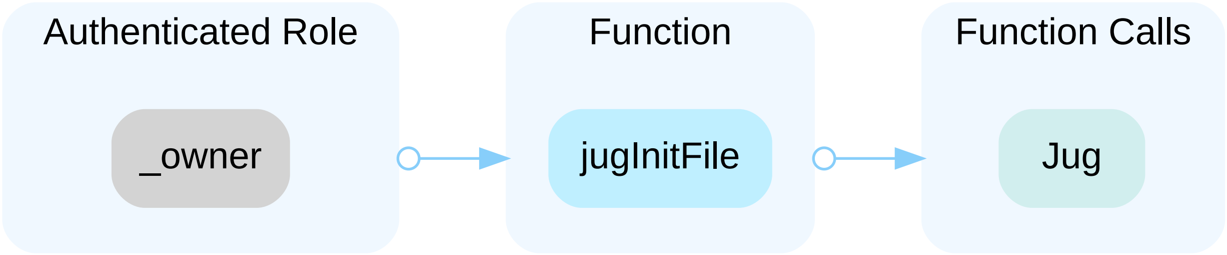
In the contract `CerosRouter` the role `_owner` has authority over the functions shown in the diagram below.



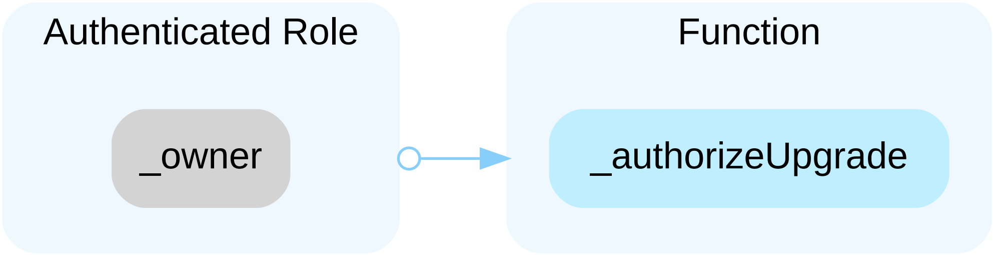
In the contract `mBNB` the role `joinContract` has authority over the functions shown in the diagram below.



In the contract `ProxyLike` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `DAOInteraction` the role `_owner` has authority over the functions shown in the diagram below.

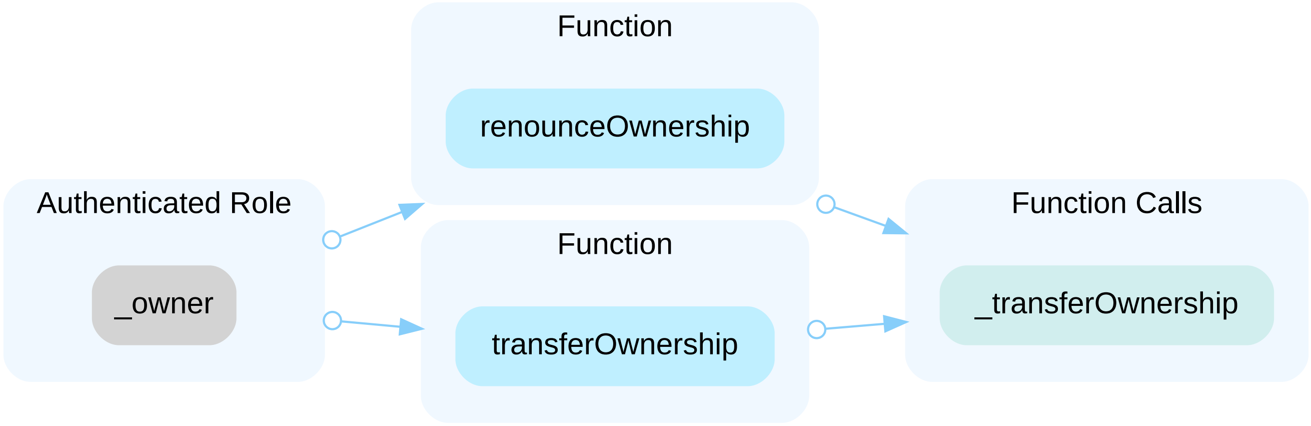


Moreover, the role `wards` has authority over following functions:

- setCores()
- setUSBApprove()
- setCollateralType()
- enableCollateralType()
- setCollateralDisc()
- removeCollateralType()

- setRewards()
- rely()
- deny()

In the contract `OwnableUpgradeable` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `HelioRewards`, the role `auth` has authority over following functions:

- stop()
- initPool()
- setHelioToken()
- setRate()
- addRewards()
- withdraw()
- deposit()
- rely()
- deny()

In the contract `HelioToken`, the role `auth` has authority over following functions:

- mint()
- burn()
- stop()
- start()

In the contract `abaci`, the role `auth` has authority over following functions:

- file()
- rely()

- deny()

In the contract `clip`, the role `auth` has authority over following functions:

- file()
- kick()
- yank()
- rely()
- deny()

In the contract `clip`, the role `auth` has authority over following functions:

- file()
- digs()
- cage()
- rely()
- deny()

In the contract `clip`, the role `auth` has authority over following functions:

- file()
- kick()
- cage()
- rely()
- deny()

In the contract `flop`, the role `auth` has authority over following functions:

- file()
- kick()
- cage()
- rely()
- deny()

In the contract `jar`, the role `auth` has authority over following functions:

- initialize()
- setSpread()
- setExitDelay()
- cage()

- rely()
- deny()

In the contract `join`, the role `auth` has authority over following functions:

- cage()
- rely()
- deny()

In the contract `join`, the role `auth` has authority over following functions:

- cage()
- rely()
- deny()

In the contract `jug`, the role `auth` has authority over following functions:

- init()
- file()
- rely()
- deny()

In the contract `pot`, the role `auth` has authority over following functions:

- cage()
- file()
- rely()
- deny()

In the contract `spot`, the role `auth` has authority over following functions:

- cage()
- file()
- rely()
- deny()

In the contract `usb`, the role `auth` has authority over following functions:

- mint()
- rely()

- deny()

In the contract `vat`, the role `auth` has authority over following functions:

- init()
- file()
- cage()
- slip()
- grab()
- suck()
- fold()
- rely()
- deny()

In the contract `vat`, the role `auth` has authority over following functions:

- fess()
- file()
- cage()
- rely()
- deny()

In the contract `CeToken`, the role `minter` has authority over following functions:

- burn()
- mint()

In the contract `CeVault`, the role `router` has authority over following functions:

- claimYieldsFor()
- withdrawFor()

In the contract `CerosRouter`, the role `router` has authority over following functions:

- claimYieldsFor()
- withdrawFor()

In the contract `HelioProvider`, the role `operator` has authority over following functions:

- claimInABNBc()

Moreover, the role `dao` has authority over following functions:

- daoBurn()
- daoMint()

Any compromise to the privileged account may allow the hacker to take advantage of this authority and update the sensitive settings and execute sensitive functions of the project.

Recommendation

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

Short Term:

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

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

Long Term:

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

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

Permanent:

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

- Renounce the ownership and never claim back the privileged roles.

  OR
- Remove the risky functionality.

CON-01 | Potential Reentrancy Attack

Category	Severity	Location	Status
Volatile Code	● Medium	contracts/clip.sol: 226, 227, 229, 264, 266, 302, 305, 400, 407, 411, 414, 418, 420, 421, 423, 424, 435, 438, 480, 481, 482; contracts/dog.sol: 221~223, 226, 231, 232; contracts/flap.sol: 156, 157, 159, 161, 162, 168, 169, 170, 181, 182; contracts/flop.sol: 151, 155, 156, 159, 162, 163, 168, 169, 180, 181; contracts/jug.sol: 130, 132, 133; contracts/vow.sol: 117, 118	ⓘ Acknowledged

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)

File: contracts/clip.sol (Line 264, Function `Clipper.kick`)

```
top = rmul(getFeedPrice(), buf);
```

- This function call executes the following external call(s).
- In `Clipper.getFeedPrice`,
  - `(pip) = spotter.ilks(ilk)`
- In `Clipper.getFeedPrice`,
  - `(val,has) = pip.peek()`
- In `Clipper.getFeedPrice`,
  - `feedPrice = rdiv(mul(uint256(val),BLN),spotter.par())`

State variables written after the call(s)

File: contracts/clip.sol (Line 266, Function `Clipper.kick`)

```
sales[id].top = top;
```

External call(s)

File: contracts/clip.sol (Line 302, Function `Clipper.redo`)

```
uint256 feedPrice = getFeedPrice();
```

- This function call executes the following external call(s).
- In `Clipper.getFeedPrice`,
  - `(pip) = spotter.ilks(ilk)`
- In `Clipper.getFeedPrice`,
  - `(val,has) = pip.peek()`
- In `Clipper.getFeedPrice`,
  - `feedPrice = rdiv(mul(uint256(val),BLN),spotter.par())`

State variables written after the call(s)

File: contracts/clip.sol (Line 305, Function `Clipper.redo`)

```
sales[id].top = top;
```

External call(s)

File: contracts/clip.sol (Line 400, Function `Clipper.take`)

```
vat.flux(ilk, address(this), who, slice);
```

File: contracts/clip.sol (Line 407, Function `Clipper.take`)

```
ClipperCallee(who).clipperCall(msg.sender, owe, slice, data);
```

File: contracts/clip.sol (Line 411, Function `Clipper.take`)

```
vat.move(msg.sender, vow, owe);
```

File: contracts/clip.sol (Line 414, Function `Clipper.take`)

```
dog_.digs(ilk, lot == 0 ? tab + owe : owe);
```

State variables written after the call(s)

File: contracts/clip.sol (Line 423, Function `Clipper.take`)

```
sales[id].tab = tab;
```

File: contracts/clip.sol (Line 424, Function `Clipper.take`)

```
sales[id].lot = lot;
```

File: contracts/clip.sol (Line 418, Function `Clipper.take`)

```
_remove(id);
```

- This function call executes the following assignment(s).
- In `Clipper._remove`,
    - `sales[_move].pos = _index`
- In `Clipper._remove`,
    - `delete sales[id]`

External call(s)

File: contracts/clip.sol (Line 400, Function `Clipper.take`)

```
vat.flux(ilk, address(this), who, slice);
```

File: contracts/clip.sol (Line 407, Function `Clipper.take`)

```
ClipperCallee(who).clipperCall(msg.sender, owe, slice, data);
```

File: contracts/clip.sol (Line 411, Function `Clipper.take`)

```
vat.move(msg.sender, vow, owe);
```

File: contracts/clip.sol (Line 420, Function `Clipper.take`)

```
vat.flux(ilk, address(this), usr, lot);
```

File: contracts/clip.sol (Line 414, Function `Clipper.take`)

```
dog_.digs(ilk, lot == 0 ? tab + owe : owe);
```

State variables written after the call(s)

File: contracts/clip.sol (Line 421, Function `Clipper.take`)

```
_remove(id);
```

- This function call executes the following assignment(s).
- In `Clipper._remove`,
  - `sales[_move].pos = _index`
- In `Clipper._remove`,
  - `delete sales[id]`

External call(s)

File: contracts/clip.sol (Line 480, Function `Clipper.yank`)

```
dog.digs(ilk, sales[id].tab);
```

File: contracts/clip.sol (Line 481, Function `Clipper.yank`)

```
vat.flux(ilk, address(this), msg.sender, sales[id].lot);
```

State variables written after the call(s)

File: contracts/clip.sol (Line 482, Function `Clipper.yank`)

```
_remove(id);
```

- This function call executes the following assignment(s).
- In `Clipper._remove`,

    - `sales[_move].pos = _index`
- In `Clipper._remove`,
    - `delete sales[id]`

External call(s)

File: contracts/dog.sol (Line 221-223, Function `Dog.bark`)

```
vat.grab(
    ilk, urn, milk.clip, address(vow), -int256(dink), -int256(dart)
);
```

File: contracts/dog.sol (Line 226, Function `Dog.bark`)

```
vow.fess(due);
```

State variables written after the call(s)

File: contracts/dog.sol (Line 231, Function `Dog.bark`)

```
Dirt = add(Dirt, tab);
```

File: contracts/dog.sol (Line 232, Function `Dog.bark`)

```
ilks[ilk].dirt = add(milk.dirt, tab);
```

External call(s)

File: contracts/flap.sol (Line 156, Function `Flapper.tend`)

```
gem.move(msg.sender, bids[id].guy, bids[id].bid);
```

State variables written after the call(s)

File: contracts/flap.sol (Line 157, Function `Flapper.tend`)

```
bids[id].guy = msg.sender;
```

External call(s)

File: contracts/flap.sol (Line 156, Function `Flapper.tend`)

```
gem.move(msg.sender, bids[id].guy, bids[id].bid);
```

File: contracts/flap.sol (Line 159, Function `Flapper.tend`)

```
gem.move(msg.sender, address(this), bid - bids[id].bid);
```

State variables written after the call(s)

File: contracts/flap.sol (Line 161, Function `Flapper.tend`)

```
bids[id].bid = bid;
```

File: contracts/flap.sol (Line 162, Function `Flapper.tend`)

```
bids[id].tic = add(uint48(block.timestamp), ttl);
```

External call(s)

File: contracts/flap.sol (Line 168, Function `Flapper.deal`)

```
vat.move(address(this), bids[id].guy, lot);
```

File: contracts/flap.sol (Line 169, Function `Flapper.deal`)

```
gem.burn(address(this), bids[id].bid);
```

State variables written after the call(s)

File: contracts/flap.sol (Line 170, Function `Flapper.deal`)

```
delete bids[id];
```

External call(s)

File: contracts/flap.sol (Line 181, Function `Flapper.yank`)

```
gem.move(address(this), bids[id].guy, bids[id].bid);
```

State variables written after the call(s)

File: contracts/flap.sol (Line 182, Function `Flapper.yank`)

```
delete bids[id];
```

External call(s)

File: contracts/flop.sol (Line 151, Function `Flopper.dent`)

```
vat.move(msg.sender, bids[id].guy, bid);
```

File: contracts/flop.sol (Line 155, Function `Flopper.dent`)

```
uint Ash = VowLike(bids[id].guy).Ash();
```

File: contracts/flop.sol (Line 156, Function `Flopper.dent`)

```
VowLike(bids[id].guy).kiss(min(bid, Ash));
```

State variables written after the call(s)

File: contracts/flop.sol (Line 159, Function `Flopper.dent`)

```
bids[id].guy = msg.sender;
```

File: contracts/flop.sol (Line 162, Function `Flopper.dent`)

```
bids[id].lot = lot;
```

File: contracts/flop.sol (Line 163, Function `Flopper.dent`)

```
bids[id].tic = add(uint48(block.timestamp), ttl);
```

External call(s)

File: contracts/flop.sol (Line 168, Function `Flopper.deal`)

```
gem.mint(bids[id].guy, bids[id].lot);
```

State variables written after the call(s)

File: contracts/flop.sol (Line 169, Function `Flopper.deal`)

```
delete bids[id];
```

External call(s)

File: contracts/flop.sol (Line 180, Function `Flopper.yank`)

```
vat.suck(vow, bids[id].guy, bids[id].bid);
```

State variables written after the call(s)

File: contracts/flop.sol (Line 181, Function `Flopper.yank`)

```
delete bids[id];
```

External call(s)

File: contracts/jug.sol (Line 130, Function `Jug.drip`)

```
(, uint prev) = vat.ilks(ilk);
```

File: contracts/jug.sol (Line 132, Function `Jug.drip`)

```
vat.fold(ilk, vow, _diff(rate, prev));
```

State variables written after the call(s)

File: contracts/jug.sol (Line 133, Function `Jug.drip`)

```
ilks[ilk].rho = block.timestamp;
```

External call(s)

File: contracts/vow.sol (Line 117, Function `Vow.file`)

```
vat.nope(address(flapper));
```

State variables written after the call(s)

File: contracts/vow.sol (Line 118, Function `Vow.file`)

```
flapper = FlapLike(data);
```

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

`[Helio]`: All of these contracts mentioned are trusted and deployed by us including the Oracle.

CON-02 | Duplication Of Code

Category	Severity	Location	Status
Coding Style, Gas Optimization	● Informational	contracts/jug.sol: 62~84; contracts/hMath.sol: 43~65	✓ Resolved

Description

The linked statements in `_rpow()`, is duplicated from the `hMath` library's `rpow` function.

Recommendation

We advise the reuse the `rpow` function from `hMath` library contract in the `jug` contract.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit.

CON-03 | Variables That Could Be Declared As Immutable

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/DssCdpManager.sol: 38; contracts/Queue.sol: 18; contracts/mBNB.sol: 9; contracts/oracle/BnbOracle.sol: 12; contracts/aBNBc.sol: 8	✓ Resolved

Description

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

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit

CRB-01 | Unchecked Variable Unit

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/ceros/CerosRouter.sol: 83, 89	✓ Resolved

Description

In the function `deposit()`, the `ratio` of `_certToken` token is used to calculate the pool's aBNBc amount `poolABNBcAmount`, and further impacts the value of staking and claiming in the pool. By default, the `ratio` is related to the unit of `_certToken`, which will be defined when deploying the `_certToken`. In order to guarantee the poolABNBcAmount calculation correctness, the unit of `ratio` must also be set to `1e18`

Recommendation

Consider adding a `require()` validator to check if the unit of `ratio` is `1e18` in the function `deposit()` in the contract `CerosRouter`.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit

CRB-02 | Missing Approval Checks On `transferFrom()` Function

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/ceros/CerosRouter.sol: 121	✓ Resolved

Description

In function `depositABNBc()`, the `_certToken.transferFrom(owner, address(this), amount);` statement did not check the approval from `owner` to `msg.sender` but relaying on the approval check in `ERC20Upgradeable.transferFrom()`. This is potentially dangerous if `ICertToken` implementation at `_certToken` does not implement a standard `ERC20Upgradeable` contract. The use case of the `depositABNBc()` function should care about permission approvals.

Recommendation

Recommend adding checks for validating whether there's enough allowance from `owner` to `msg.sender`. The `allowance()` function in `ERC20Upgradeable.sol` could be a reference:

https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/token/ERC20/ERC20Upgradeable.sol#L127

Alleviation

`[Helio]`: CertToken is implemented ERC20 standard and we can receive this type of error from it.

CRB-03 | Inconsistent Comment And Code

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/ceros/CerosRouter.sol: 127~128	✓ Resolved

Description

The comment states `// let's check balance of CeRouter in aBNBc`, however the function does not check the balance but transfers the `_certToken` token from `owner` to `address(this)`.

Recommendation

Check the code to determine whether aBNBc balance check should be implemented, or update the comment to improve clarity and avoid confusion.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit.

DAO-01 | Duplicated Validation

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/DAOInteraction.sol: 225, 262, 327	✓ Resolved

Description

```
require(collateralType.live == 1, "Interaction/inactive collateral");
```

collateral type is validated twice in the function `deposit()` and `borrow()`, in which `collateralType.live` checked in these functions and invocation of `drip()`.

Recommendation

Consider remove the below require validation in L225 and L262

```
require(collateralType.live == 1, "Interaction/inactive collateral");
```

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit

HPB-01 | Typo

Category	Severity	Location	Status
Coding Style	● Informational	contracts/ceros/HelioProvider.sol: 138, 140	✓ Resolved

Description

`minumunUnstake` is misspelled

Recommendation

Consider fixing `minumunUnstake` to `minimumUnstake`

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit.

HRB-01 | Duplication Of Code

Category	Severity	Location	Status
Coding Style, Gas Optimization	● Minor	contracts/HelioRewards.sol: 134~141	✓ Resolved

Description

The `withdraw()` code block is equivalent to the `deposit()` function.

Recommendation

We recommend the team to remove one of the two functions to to increase the reusability. If the functionality of `withdraw()` and `deposit()` differ in the original design, then we advice the team to revisit the functions implementations

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit.

HRB-02 | Usage Of Magic Number

Category	Severity	Location	Status
Magic Numbers	● Informational	contracts/HelioRewards.sol: 116	✓ Resolved

Description

The `distributionApy()` function uses the magic number `31536000` in calculating the APY. The value will not be updated if constants are modified.

Recommendation

We recommend replacing the magic number `31536000` with `365 * 24 * 3600`

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit

JAR-01 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/jar.sol: 194	✓ Resolved

Description

According to the current implementation, the value `unstakeTime` of each `account` is 0 by default until this account `exit()` the contract. There could be a validation to check if any given `account` address's `unstakeTime` is 0.

Recommendation

Consider add `require(unstakeTime[accounts[i] != 0, "unstake ")` to for loop of the `redeemBatch()` function.

```
if (block.timestamp < unstakeTime[accounts[i]] && unstakeTime[accounts[i]] != 0)
```

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit.

USB-01 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	● Informational	contracts/usb.sol: 53, 56, 71	✓ Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the latest commit.

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.

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.

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.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

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.