A CONSENSYS DILIGENCE AUDIT REPORT

Fairmint Continuous Securities Offering

Date	November 2019

1 Summary

- **Project Name:** C-Org Audit
- **Client Name:** Fairmint
- **Client Contacts:** Thibauld Favre, Nick Cuso
- **Lead Auditor:** John Mardlin
- **Co-auditors:** Daniel Luca, Neil McLaren, Alexander Wade

1.1 Scope

- **Repository:** https://github.com/Fairmint/c-org
- **Git commit hashes:**
  - **Initial review:** 14f4e3e02b1d756d4d3caad34fbed07a9c0f09a1
  - **Mitigations review:** 060ea3f55aa10012831c9a0c63c61d5dd9ebf3a3
- **Documentation:**
  1. https://github.com/Fairmint/c-org/wiki
  2. https://github.com/c-org/whitepaper

The audit team evaluated the system's security, resilience, and looked for unexpected behaviors relative to its specification. The audit activities can be grouped into the following three broad categories:

1. **Security:** Identifying security related issues within the contract.

2. **Architecture:** Evaluating the system architecture through the lens of established smart contract best practices.
3. **Code quality:** A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Scalability
    - Code complexity
    - Quality of test coverage

2 Timeline

Our review was split into two phases as outlined below:

2.1 Initial review

The initial review period took place from August 26, 2019 to September 9th, 2019.

During the initial review period the following Vyper files were covered:

File	SHA-1 hash
BigDiv.vy	5a8d7f35bcf828de878b225760134bdeb42b3339
DecentralizedAutonomousTrust.vy	a4bf4965862892a112f54bc038cd46e154046015
ERC1404.vy	0a0acbffa496c5e0df524fb131509defe67ed7f6
FAIR.vy	14224094b972494793455ab91609443c02394ce8

During this period, we focused on identifying a wide range of known smart contract vulnerabilities. In addition we sought to verify that:

- The large number mathematics in BigDiv were within the acceptable error range
- The mathematics of the bonding curve model were sound
- Funds controlled by the DecentralizedAutonomousTrust, in the form of ETH, ERC20, or ERC777 could not be withdrawn by unauthorized parties.

2.2 Mitigations review

Following the initial review, as well as our subsequent security review of the Vyper compiler itself, the Fairmint team elected to rewrite the contracts in Solidity. We then performed a one-week review of the updated code from November 25th to November 29th.

File	SHA-1 hash
contracts/DecentralizedAutonomousTrust.sol	d3034bcc1ced07760a3500a5777dec614a562f83
contracts/Whitelist.sol	c376b6997489abb7adb92357a53a976a72c05e3c
contracts/math/BigDiv.sol	2ebc41e4b220c4a4dacacc2bfced9b884c3b0976
contracts/math/Sqrt.sol	ce75fd87ac1b5f597dcc348679c38ac1401aa67e

During this phase we focused on verifying the following aspects of the codebase:

- the mathematics of BigDiv.sol and Sqrt.sol
- the SafeMath library was used in any situation where arithmetic operations posed a risk of overflow or underflow.

Given the very thorough test suite, and that the Solidity was a direct port from the audited Vyper implementation with very minimal modifications (mostly reduced complexity), we placed less emphasis on identifying logic errors or failure to correctly implement the specification.

3 Recommendations

During the course of our review, we made the following recommendations:

3.1 Avoid Vyper [Done]

The system was originally written in Vyper, which its own developers refer to as an “experimental language”, still in beta, and undergoing active development. During the audit process, the audit team identified and reported several bugs in the compiler. Another issue was reported by the client. Our subsequent review of the compiler itself recommended major architectural changes.

We ultimately recommended porting the code to Solidity.

3.2 Remove ERC-777 [Done]

Support 3 currency types as investment (ERC-20, ERC-777, and ETH), and 2 currency types as a security token (ERC-20, ERC-777) introduces a significant number of branches in the code and potential for logical bugs.

We recommended removing ERC-777 if not deemed absolutely necessary.

3.3 Don’t repeat yourself [Done]

`_toDecimalWithPlaces` (in the Vyper implementation) is tricky to reason about in the context it which it is called, which is always as part of the same series of operations designed to calculate an integer square root. It may make sense to rename it as integerSquareRoot and move some of the surrounding code into the function.

3.4 Fixes [Done]

Address issues we’ve listed with a severity of Medium or greater. For Minor issues, consider which can be addressed with minimal effort and new code.

3.5 Prepare for future compatibility with GAS repricing EIPs [Done]

Future upgrades to the opcode prices in the EVM may cause some contracts to fail. We are now recommending using `raw_call`.

3.6 Thoroughly document and comment BigDiv [Done]

BigDiv is a large and complex contract. Future developers and reviewers would benefit from having it much more thoroughly commented.

3.7 Review the Code Quality suggestions [Done]

Section 6 of this report compiles suggestions which do not pose a direct threat to security, but would otherwise improve the quality of the code. Those suggestions should be considered.

3.8 Considerations for Upgradability

When deployed with proxies as planned, the contract logic could theoretically be re-written to confiscate currency and tokens arbitrarily.

When deployed without proxies, the Ethereum accounts `DAC.control` and `ERC1404.owner` both have the ability to lock (but not confiscate) user funds indefinitely.

We recommend having a documented and well tested process for upgrading the contract logic by changing the configuration of each proxy. The testing should include checks that the contract's storage layout is backwards compatible with previous versions.

4 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

4.1 Actors

The relevant actors are as follows:

- **Beneficiary / C-Org**: Deploys and initializes the DAT, FAIR, and BigDiv contracts. If the beneficiary address buys into the DAT via `buy`, all funds sent are directed to the DAT's buyback reserve. The beneficiary is expected to direct some portion of revenue through the DAT contract, in order to reward FAIR holders.
- **DAT Controller**: This is initially the C-Org itself, but is updateable (by the C-Org) to any nonzero address. The control address can, at any time, change several properties in both the DAT and FAIR contracts (see details in Trust Model).
- **Investor**: The investor buys into the DAT through its `buy` function. In return, they receive FAIR tokens. A portion of their purchase goes to the DAT's buyback reserve.
- **Customer**: Customers can pay the C-Org via the DAT's `pay` function. A portion of payment is sent to the buyback reserve, and the rest is sent to the beneficiary address.

4.2 Trust Model

In any smart contract system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- The beneficiary is represented by a single address. If this address is a multisig, it is expected that this multisig correctly handles interactions with the DAT contract system. If this address is an EOA, it is expected that the account's private key is not compromised.

- The beneficiary is expected not to use a different address to buy FAIR from the DAT.
- The control address can change several properties at any time:
  - The ERC1404 address, which may hold important transfer restrictions, is expected to implement restrictions in line with investor expectations.
  - The beneficiary address, which represents the entire C-Org on-chain, is expected to effectively represent the organization.
  - The control address, which can change these properties, is expected to be used in a manner according to investor expectations.
  - The fee collection address, which receives fees from investment into the DAT, is expected not to revert. If the fee collection address reverts when it receives Ether, investors will be unable to buy into the DAT.
  - The fee basis points are expected to remain consistent with investor expectation, given that a fee basis equal to 10000 will result in the entire buyback reserve being sent to the fee collection address on DAT buy-in.
  - The minimum investment amount is expected to remain consistent with investor expectations.
  - The time until the DAT can be closed (`openUntilAtLeast`) is expected to remain consistent with investor expectations.

5 Verification of Custom Properties

5.1 Custom properties

The following properties were specifically verified using automated techniques.

Property	High-level Description	Result
Sqrt.sol: Square root calculation	The square root of a number is correct to within 1 (the minimum error based on integer math restrictions).	✔
BigDiv.sol: Division of large integers	The result of calling `BigDiv2x1()` is within the allowable error range of the decimal result.	✔

5.2 Verification Methods

- **Sqrt.sol:** In order to verify the math in Sqrt.sol, we created two custom checks using Solidity's `assert()` function. Using the MythX security analysis engine, we

then verified that the code could not be executed in such as away as to invalidate the assert statements.

- **BigDiv.sol:** In order to verify the math in BigDiv.sol, we used a method closer to 'brute force verification'.
  1. We defined a large array of input numbers, including potentially interesting values.
  2. We then ran a test for all possible combinations of numbers. The test compared the value returned by the `BigDiv` methods with the value as calculated with decimal math using `bigNumber.js`. The test passes if the two values are within 0.000001%.
  3. This method was subsequently expanded to include even more input numbers, giving us high confidence in the correctness

6 Issues

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 FAIR can be stolen using ERC-777 hooks Critical ✓ Fixed

Resolution
fixed by completely removing ERC-777 support.

Description

The `sell()` function calls out to user-configured hooks when burning incoming FAIR tokens. The `buy()` function does the same if the DAT's currency is ERC-777 compliant.

Either of these hooks might invoke malicious code to re-enter the DAT, allowing them to sell and/or buy FAIR tokens at an unintentionally favourable price.

Such attacks may leave the DAT undercollateralized, resulting in other investors being unable to redeem their FAIR for currency.

Example

Here are some ordered extracts from the code invoked when `DAT.buy()` is called, when the DAT's currency is an ERC-777 compliant token.

code/contracts/DecentralizedAutonomousTrust.vy:L629

```
tokenValue: uint256 = self.estimateBuyValue(_currencyValue)
```

The code above does a calculation using `FAIR.totalSupply` as input. The higher `FAIR.totalSupply` is, the more expensive FAIR tokens become.

code/contracts/DecentralizedAutonomousTrust.vy:L502-L503

```
if(self.isCurrencyERC777):
  self.currency.operatorSend(_from, self, _quantityToInvest, "", "")
```

Per the ERC-777 standard, the code above invokes an arbitrary `tokensToSend()` hook configured by the buyer.

code/contracts/DecentralizedAutonomousTrust.vy:L654

```
self.fair.mint(msg.sender, _to, tokenValue, "", "")
```

The code above increments `FAIR.totalSupply`, effectively increasing the price of FAIR tokens. This happens *after* the other two code extracts have completed.

An attacker can exploit re-entrancy during the `tokensToSend()` hook, to purchase further tokens at a (perhaps extremely) favourable price *before* `FAIR.totalSupply` is incremented.

If the price at the time of the initial `buy()` is very low (as it will be when `totalSupply` is small or zero), then they may be able to buy huge amounts of FAIR at that very low price.

Recommendation

Prevent reentrancy by adding a mutex (using Vyper's `@nonreentrant()` decorator) across all functions that result in ERC-777 token transfers (of FAIR or an ERC-777 `currency`).

6.2 BigDiv does not prevent overflow in some cases where it should Medium ✓ Fixed

Resolution
Fixed in the Solidity implementation.

Description

`BigDiv.vy` has been created with the aim of allowing calculations like `(a * b) / d` to succeed where an intermediate step (e.g. `a * b`) might overflow but the end result is `<= MAX_UINT256`.

All of the functions sometimes fail in this aim if the numerators are large and of the same order of magnitude. (E.g. for `bigDiv2x1`, it fails if `_numA / MAX_BEFORE_SQUARE = numB / MAX_BEFORE_SQUARE > 0`)

The chances of this issue being hit accidentally or exploited deliberately in the current code will both greatly depend on the DAT's configuration and its state. (If the numbers are amenable, an attacker could conceivably front run transactions and adjust FAIR balances in a way that causes targeted transactions to fail.)

Having functions that unexpectedly fail is dangerous for future consumers of this code, and the (simplest possible) fix is small.

Examples

The following code overflows in the code as audited, but succeeds (returning `MAX_INT`) if `MAX_BEFORE_SQUARE` is altered as suggested in issue 6.4.

```
a='340282366920938463463374607431768211455'
b='340282366920938463463374607431768211457'
BigDiv.bigDiv2x1(a, b, '1', false) -- ???
```

`bigDiv2x1` also overflows for some simple cases where the result is *far* below `MAX_UINT256`. E.g.:

```
a='340282366920938463463374607431768211456'
BigDiv.bigDiv2x1(a, a, a, false) -- overflows, despite result being ~sqrt(MAX_INT)
```

Recommendations

1. Fix overflows

The following code appears in each `BigDiv` function:

code/contracts/BigDiv.vy:L30-L31

```
if(factor == 0):
  factor = 1
```

Replacing every instance of these two lines with simply `factor += 1` will avoid overflows. It will also reduce the (currently undocumented) accuracy of the result in some cases, so see recommendations in issue 6.4.

2. Add automated regression tests for all `BigDiv` functions

We have already written some basic test code and can supply it on request.

6.3 Square roots are not calculated accurately for inputs below ~10^30 Minor ✓ Fixed

Resolution
This has been addressed.

Description

The rounding performed when calculating square roots results in an extreme loss of precision for numbers < ~10^30.

This may or may not be OK. Typically the numbers being square rooted will be significantly larger than 10^30, but when supply is low and the value of a buy / pay is also low, this rounding could have a dramatic effect.

In any case, the square rooting logic and its limitations could be be better documented and tested.

Examples

In both places where `_toDecimalWithPlaces` is used, it is surrounded by the same code, which combines with `_toDecimalWithPlaces` to calculate a square root of a `uint256`:

code/contracts/DecentralizedAutonomousTrust.vy:L792-L808

```
# Math: Truncates last 18 digits from tokenValue here
tokenValue /= DIGITS_UINT

# Math: Truncates another 8 digits from tokenValue (losing 26 digits in total)
# This will cause small values to round to 0 tokens for the payment (the payment is sti
# Math: Max supported tokenValue is 1.7e+56. If supply is at the hard-cap tokenValue wo
# for a _currencyValue up to 1.7e33 (or 1.7e15 after decimals)
decimalValue: decimal = self._toDecimalWithPlaces(tokenValue)

decimalValue = sqrt(decimalValue)

# Unshift results
# Math: decimalValue has a max value of 2^127 - 1 which after sqrt can always be multip
# here without overflow
decimalValue *= DIGITS_DECIMAL

tokenValue = convert(decimalValue, uint256)
```

This code casts the number to a `decimal` so that Vyper's `sqrt` can be used, after first doing some rounding to prevent overflow during the cast. After all of this is done, it casts back to a `uint256`.

The result of the rounding + casts is reasonably accurate square root for very large integers, but it loses a lot of accuracy for smaller integers.

E.g. an integer as "small" as `123456789012345678901234 56` results in a "square root" value of `0`.

Recommendations

1. Reduce code duplication and document assumption / limitations

By moving the common surrounding code *inside* the `_toDecimalWithPlaces` function, that function could be renamed (e.g. to `integerSqrt`) and the limitations of the whole square root calculation could be more easily documented.

2. Test the documented limitations of the `integerSqrt` operation

To verify the documented limitations, thereby reducing the chances of this code being misused by a different developer at a later stage of the same project.

3. *If* accuracy for smaller integers is important, improve it

Greater accuracy may be achievable by writing or importing a function that approximates square roots using integer arithmetic and Newton's Method, without ever casting to a `decimal`.

6.4 BigDiv estimates some values that could be easily calculated

Minor ✓ Fixed

Resolution
Fixed in the port to Solidity.

Description

The accuracy of `BigDiv`'s functions is neither documented clearly nor directly tested. (The csv tests should exercise much of the `BigDiv` code, but it's hard to see exactly what calculations are being done.)

`BigDiv` returns estimates in some cases where it could easily calculate a precise answer. Having spoken offline about FAIR's requirements, we believe the lack of accuracy itself is probably not a problem right now, but it creates a small risk of `BigDiv` being accidentally misused in future scenarios where its level of accuracy is insufficient (perhaps by a different developer, during a new phase of the FAIR project).

In any case, `BigDiv`'s behaviour could be better documented and tested.

Examples

For comparison, we define a simpler function:

```
@public
@constant
def simpleDiv(
  _numA: uint256,
  _numB: uint256,
  _den: uint256
) -> uint256:
  return _numA * _numB / _den
```

In some cases where both `bigDiv2x1` and `simpleDiv` both succeed, `bigDiv2x1` is less accurate than `simpleDiv`:

```
a='1'
b='999934028236692093846346337460743176821145799999'
BigDiv.bigDiv2x1(a, b, '8', false) -- succeeds, approximate answer
simpleDiv(a, b, '8')               -- succeeds, exact answer
```

Also, the constants `MAX_BEFORE_SQUARE` and `MAX_BEFORE_CUBE` seem to have been miscalculated, resulting in estimation happening slightly more often than necessary.

Recommendations

1. Document expected accuracy / rounding

To prevent accidental misuse of these functions in the future.

2. Add automated regression tests for all `BigDiv` functions

We have written some basic unit test code as part of our audit, and can supply it on request.

3. *If* maximising accuracy is important, improve it

There is some low-hanging fruit here, such as:

- increasing `MAX_BEFORE_SQUARE` and `MAX_BEFORE_CUBE` to 340282366920938463463374607431768211456 and 48740834812604276470692695, respectively.
  - Per code logic, these numbers are really the first numbers that *cannot* be squared and cubed, so you may also wish to rename the constants.
  - Note that `MAX_BEFORE_SQUARE` is also defined in the DAT contract
- Add a check for overflow before resorting to estimation. E.g. for `bigDiv2x...`:

```
if(MAX_UINT256 / _numA > _numB):
    # No rounding required. Return exact result
    return _numA * _numB / _den
```

This latter change may reduce gas consumption as well as improving accuracy.

6.5 Unused code in `BigDiv` functions Minor ✓ Fixed

Resolution

Fixed.

Description

Some parameters and associated logic can be removed from `BigDiv`'s functions. This would simplify the code, as well as the analysis and testing of the code.

Examples

The `_roundUp` parameter is always `false` in the following functions:

- `bigDiv2x1`
- `bigDiv3x1`
- `bigDiv3x3`

Associated conditionals are numerous. E.g. `bigDiv3x3`'s code branches 7 times on the value of `_roundUp`, even though it is always false.

Recommendation

Remove unused code and associated logic.

Add tests for code that remains.

6.6 FAIR - Calling `transferFrom` should not emit the `Approval` event Minor Won't Fix

Resolution

Closed as WontFix.

This behavior is a de facto standard based on it's usage in the OpenZeppelin implementation of ERC20.

Description

The method `transferFrom()` sends some already approved tokens to some address:

code/contracts/FAIR.vy:L427-L439

```
@public
def transferFrom(
  _from: address,
  _to: address,
  _value: uint256
) -> bool:
  """
  @notice Transfers `_value` amount of tokens from address `_from` to address `_to` i
  """
  self.allowances[_from][msg.sender] -= _value
  self._send(msg.sender, _from, _to, _value, False, "", "")
  log.Approval(_from, msg.sender, self.allowances[_from][msg.sender])
  return True
```

But it also emits an `Approval` event.

code/contracts/FAIR.vy:L438

```
log.Approval(_from, msg.sender, self.allowances[_from][msg.sender])
```

The event does not seem to create problems, it basically updates the remaining approved tokens.

Examples

The EIP 20 documentation states that the event should be emitted when a successful call to `approve` happens. It does not say if it should (or should not) be used when successfully calling `transferFrom()`.

https://eips.ethereum.org/EIPS/eip-20#approval

It does not seem to violate the EIP 20 or EIP 777 standard and it helps any off-chain service monitoring the contract, keep track of how many remaining approved tokens are left, without having any previous state.

However any re-entrancy issues will make the transaction emit multiple events, each event having different amounts approved, the last emitted event having the highest value, which will be the incorrect one.

Recommendation

We suggest removing the emitted log because it can create problems.

6.7 On-chain logic cannot reliably prevent a malicious beneficiary from purchasing tokens at a discount Minor ✓ Fixed

Resolution

closed as WontFix. Fraudulent token purchases by the Beneficiary are prevented by the associated legal agreements, not by on-chain logic.

The extra code should actually be thought of as enabling a legitimate method for the Beneficiary purchase tokens at a fair price.

Description

The `buy()` function contains unique logic for identifying and processing an investment by the beneficiary:

code/contracts/DecentralizedAutonomousTrust.vy:L650-L658

```
elif(self.state == STATE_RUN):
  if(_to != self.beneficiary):
    self._distributeInvestment(_currencyValue)

self.fair.mint(msg.sender, _to, tokenValue, "", "")

if(self.state == STATE_RUN):
  if(_to == self.beneficiary):
    self._applyBurnThreshold() # must mint before this call
```

Because the beneficiary receives a portion amount invested, without this logic the beneficiary organization could purchase FAIRs for a fraction of the price compared to external investors.

However, this logic can be easily circumvented by a dishonest beneficiary using another address for investments.

The Fairmint team has explained that they are aware that this protection can be circumvented. The “legal layer” is necessary to enforce good behaviour, and the

beneficiary would be committing fraud in case they purchased FAIRs using another address. Thus the extra code should actually be thought of as enabling the Beneficiary to legitimately purchase tokens.

Recommendation

This functionality introduces extra code. Consider reducing complexity by removing this functionality if it is not essential.

6.8 FAIR is not ERC-777 compliant Minor ✓ Fixed

Resolution
fixed by removing ERC-777 support

Description

A comment at the top of `FAIR.vy` describes it as an "ERC-777 and ERC-20 compliant token".

But by the code's own acknowledgement, it is not *fully* ERC-777 compliant in its current state.

Examples

The contract is non-compliant with ERC-777 in at least the following ways:

- Does not allow per-user revocation of the default operator (the DAT)
- Does not call the ERC777 `tokensToSend` and `tokensReceived` hooks within `transfer` and `transferFrom`
- It is (correctly, given the points above) not ERC820-registered as an `ERC777Token`

This list may not be exhaustive.

Recommendation

Implementing the standard fully may improve interop, so implementing all missing logic should be considered.

If not in full compliance:

1. avoid publishing any documentation that could be construed as claiming ERC-777 compliance, including code comments.
2. in accordance with other findings, consider removing ERC-777 compliance, and restricting the functionality to ERC-20.

6.9 Not compliant with ERC1404 Minor ✓ Fixed

Resolution
Fixed. Porting to solidity enabled compliance with ERC1404.

Description

The ERC1404 standard is an extension of the ERC20 standard. Here it has been implemented as a standalone contract, but does not contain all of the extra functions required by ERC1404.

As such, neither the `FAIR` contract nor the `ERC1404` contract is ERC1404-compliant.

Recommendations

1. Rename the `ERC1404` contract to be something more generic like `Whitelist`. This is more descriptive, and avoids confusion between `Whitelist.approve()` and the completely unrelated `approve()` function that an ERC1404-compliant contract should inherit from ERC20.
2. Fully implement ERC1404 in `FAIR` by adding `messageForTransferRestriction()`, if and only if the standard can be changed to accommodate Vyper's types. If it cannot, drop all claims or implications of ERC1404 support.
3. To further reduce confusion, consider renaming `approve()`, and perhaps splitting it into 2 separate functions. E.g. `allow()` and `deny()`.

7 Code quality recommendations

This sections compiles suggestions which do not pose a direct threat to security, but would otherwise improve the quality of the code.

7.1 DecentralizedAutonomousTrust.sol

The method `_authorizeTransfer` could be rewritten as a modifier, if desired.

7.2 Whitelist.sol

The argument `_isSell` is not used in the method `authorizeTransfer()`.

7.3 Sqrt.sol

`SafeMath.sol` is imported to `Sqrt.sol`, but is not used.

8 Gas efficiency optimization recommendations

8.1 DecentralizedAutonomousTrust.sol

The `BigDiv.sol` and `Sqrt.sol` contracts are deployed separately and their methods are accessed as external calls. This is more expensive than accessing the functions as libraries. Ie. `library BigDiv` and `using BigDiv for uint`.

Appendix 1 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

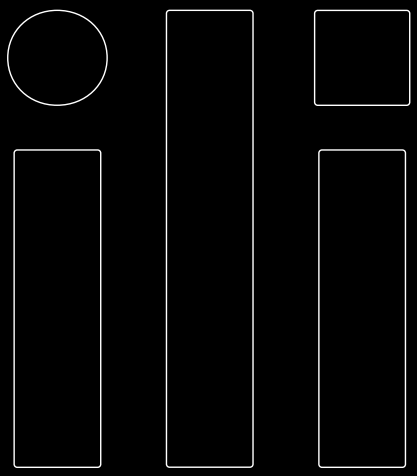
PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.

CONTACT US



AUDITS

FUZZING

SCRIBBLE

BLOG

TOOLS

RESEARCH

ABOUT

CONTACT

CAREERS

PRIVACY POLICY

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

Email*

e-mail address

→