Zer0 - zDAO Token

Date	May 2021
Lead Auditor	David Oz Kashi
Co-auditors	Martin Ortner

1 Executive Summary

This report is part of a series of reports presenting the results of our engagement with **zer0** to review **zNS, zAuction, and zBanc, zDAO Token**.

The review was conducted over four weeks, from **19 April 2021** to **21 May 2021**. A total of 2x4 person-weeks were spent.

1.1 Layout

It was requested to present the results for the four code-bases under review in individual reports. Links to the individual reports can be found below.

The Executive Summary and Scope sections are shared amongst the individual reports. They provide a general overview of the engagement and summarize scope changes and insights into how time was spent during the audit. The section Recommendations and Findings list the respective findings for the component under review.

The following reports were delivered:

- zNS

- zAuction
- zBanc
- zDAO-Token

1.2 Assessment Log

In the first week, the assessment team focussed its work on the `zNS` and `zAuction` systems. Details on the scope for the components was set by the client and can be found in the next section. A walkthrough session for the systems in scope was requested, to understand the fundamental design decisions of the system as some details were not found in the specification/documentation. Initial security findings were also shared with the client during this session. It was agreed to deliver a preliminary report sharing details of the findings during the end-of-week sync-up. This sync-up is also used to set the focus/scope for the next week.

In the second week, the assessment team focussed its work on `zBanc` a modification of the bancor protocol solidity contracts. The initial code revision under audit (zBanc `48da0ac1eebbe31a74742f1ae4281b156f03a4bc`) was updated half-way into the week on Wednesday to zBanc (`3d6943e82c167c1ae90fb437f9e3ed1a7a7a94c4`). Preliminary findings were shared during a sync-up discussing the changing codebase under review. Thursday morning the client reported that work on the `zDAO Token` finished and it was requested to put it in scope for this week as the token is meant to be used soon. The assessment team agreed to have a brief look at the codebase, reporting any obvious security issues at best effort until the end-of-week sync-up meeting (1day). Due to the very limited left until the weekly sync-up meeting, it was recommended to extend the review into next week as. Finally it was agreed to update and deliver the preliminary report sharing details of the findings during the end-of-week sync-up. This sync-up is also used to set the focus/scope for the next week.

In the third week, the assessment team continued working on `zDAO Token` on Monday. We provided a heads-up that the snapshot functionality of zDAO Token was not working the same day. On Tuesday focus shifted towards reviewing changes to zAuction (`135b2aaddcfc70775fd1916518c2cc05106621ec`, remarks). On the e day the client provided an updated review commit for zDAO Token (`81946d451e8a9962b0c0d6fc8222313ec115cd53`) addressing the issue we reported on

Monday. The client provided an updated review commit for zNS (`ab7d62a7b8d51b04abea895e241245674a640fc1`) on Wednesday and zNS (`bc5fea725f84ae4025f5fb1a9f03fb7e9926859a`) on Thursday.

As can be inferred from this timeline various parts of the codebases were undergoing changes while the review was performed which introduces inefficiencies and may have an impact on the review quality (reviewing frozen codebase vs. moving target). As discussed with the client we highly recommend to plan ahead for security activities, create a dedicated role that coordinates security on the team, and optimize the software development lifecycle to explicitly include security activities and key milestones, ensuring that code is frozen, quality tested, and security review readiness is established ahead of any security activities. It should also be noted that code-style and quality varies a lot for the different repositories under review which might suggest that there is a need to better anchor secure development practices in the development lifecycle.

After a one-week hiatus the assessment team continued reviewing the changes for `zAuction` and `zBanc` . The findings were initially provided with one combined report and per client request split into four individual reports.

2 Scope

Our review focused on the following components and code revisions:

2.1 Objectives

Together with the zer0 team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

2.2 Week - 1

- zNS (`b05e503ea1ee87dbe62b1d58426aaa518068e395`) (scope doc) (1, 2)
- zAuction (`50d3b6ce6d7ee00e7181d5b2a9a2eedcdd3fdb72`) (scope doc) (1, 2)

Original Scope overview document

2.3 Week - 2

- zBanc (`48da0ac1eebbe31a74742f1ae4281b156f03a4bc`) initial commit under review
- zBanc (`3d6943e82c167c1ae90fb437f9e3ed1a7a7a94c4`) updated commit under review (mid of week) (scope doc) (1)
  - Files in Scope:
    - `contracts/converter/types/dynamic-liquid-token/DynamicLiquidTokenConverter`
    - `contracts/converter/types/dynamic-liquid-token/DynamicLiquidTokenConverterFactory`
    - `contracts/converter/ConverterUpgrader.sol` (added handling new converterType 3)
- zDAO token provided on thursday (scope doc) (1)
  - Files in Scope:
    - `ZeroDAOToken.sol`
    - `MerkleTokenAirdrop.sol`
    - `MerkleTokenVesting.sol`
    - `MerkleDistributor.sol`
    - `TokenVesting.sol`
    - And any relevant Interfaces / base contracts

The `zDAO` review in week two was performed best effort from Thursday to Friday attempting to surface any obvious issues until the end-of-week sync-up meeting.

2.4 Week - 3

- Continuing on zDAO token (`1b678cb3fc4a8d2ff3ef2d9c5625dff91f6054f6`)
- Updated review commit for zAuction (`135b2aaddcfc70775fd1916518c2cc05106621ec` , 1) on Monday
- Updated review commit for zDAO Token (`81946d451e8a9962b0c0d6fc8222313ec115cd53`) on Tuesday

- Updated review commit for zNS (`ab7d62a7b8d51b04abea895e241245674a640fc1`) on Wednesday
- Updated review commit for zNS (`bc5fea725f84ae4025f5fb1a9f03fb7e9926859a`) on Thursday

2.5 Hiatus - 1 Week

The assessment continues for a final week after a one-week long hiatus.

2.6 Week - 4

- Updated review commit for zAuction (`2f92aa1c9cd0c53ec046340d35152460a5fe7dd0` , 1)
- Updated review commit for zAuction addressing our remarks
- Updated review commit for zBanc (`ff3d91390099a4f729fe50c846485589de4f8173` , 1)

3 Update: `23 Aug 2021` - `WILD` Token

On `20 Aug 2021` the client requested the inclusion of a deployed instance of the `zDAOToken` named `WILD` to this report. The `WILD` token at 0x2a3bff78b79a009976eea096a51a948a3dc00e34 (proxy) (implementation) is a slightly modified variant of the `zDAOToken` initially under review. A new function `initializeImplementation()` has been added for the purpose of initializing the implementation after deployment.

```
⇒  diff zDAO-Token/contracts/ZeroDAOToken.sol deployed.sol --unified
--- zDAO-Token/contracts/ZeroDAOToken.sol       2021-05-19 14:35:42.000000000 +
+++ deployed.sol        2021-08-23 13:44:09.000000000 +0200
@@ -31,6 +31,12 @@
     __ERC20Pausable_init();
   }

+  // Call this on the implementation contract (not the proxy)
+  function initializeImplementation() public initializer {
+    __Ownable_init();
+    _pause();
+  }
+
   /**
    * Mints new tokens.
    * @param account the account to mint the tokens for
```

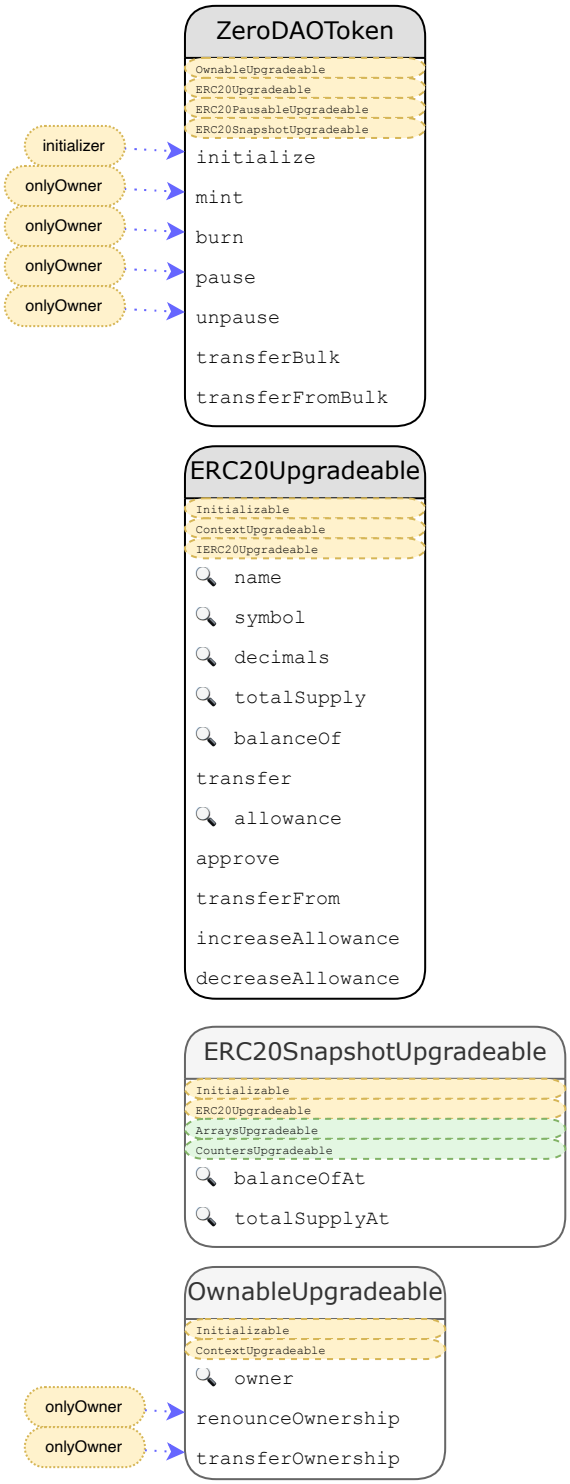
We would like to note that instead of having a dedicated new method `initializeImplementation()` it would be more natural to enforce an initialization and `pause()` in the `constructor` instead. This way the implementation gets paused immediately at deployment and no other - potentially front-runnable - transaction is required. It should further be noted that the `AdminUpgradeabilityProxy` contract deployed at 0x2a3bff78b79a009976eea096a51a948a3dc00e34 was **not** in scope of this review. A bytecode verification was **not** performed. The deployed contract lexically matches the upstream `ZeroDAOToken`.

4 System Overview

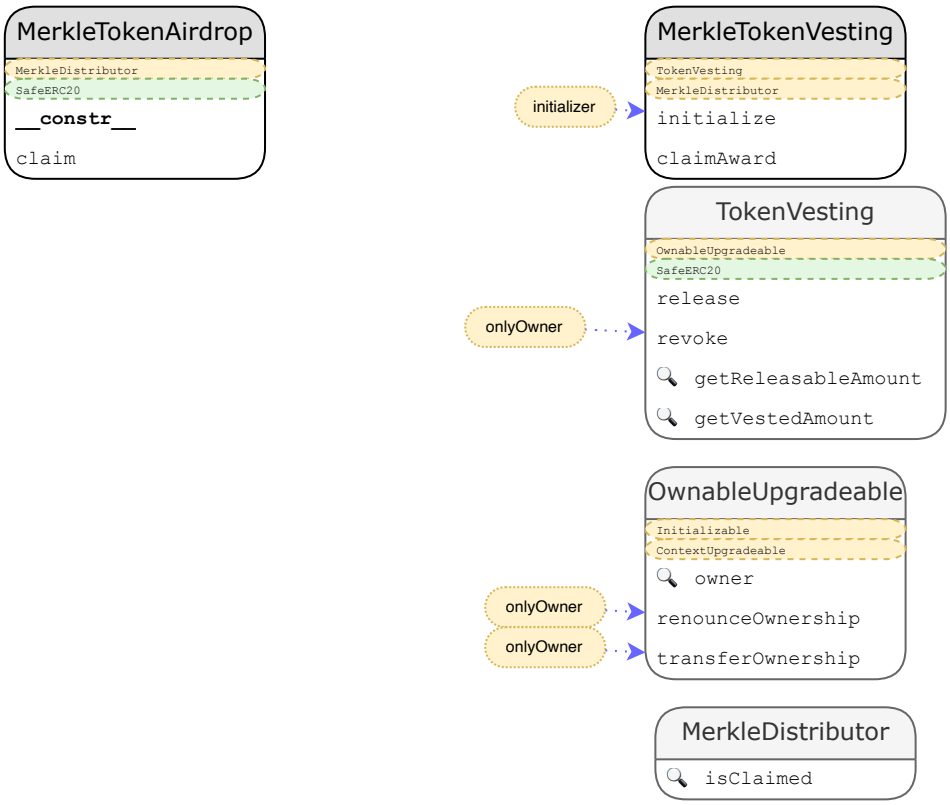
This section describes the top-level/deployable contracts, their inheritance structure and interfaces, actors, permissions and important contract interactions of the initial system under review. This section does not take any fundamental changes into account that were introduced during or after the review was conducted.

Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The 🔍 icon indicates that a method is declared as non-state-changing (view/pure) while other methods may change state. A yellow lined row at the top of the contract shows inherited contracts. A green

dashed row at the top of the contract indicates that that contract is used in a usingFor declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.



zDAO Token



zDAO Merkledrop/Vesting

The zDAO Token specification can be found in the project's repository. The system is comprised of three main components:

- zDAO Token
- Merkle Tree Token Vesting
- Merkle Tree Airdrop

The token is `mintable`, `burnable`, and `pausable`. The owner of the contract has wide-ranging power of the token and can mint/burn and pause at will. It is, therefore, important for users to verify that the token is owned by the zero DAO as mentioned in the specification:

> The owner of the token contract has the ability to mint and burn tokens. The intended owner of this token is a Zer0 DAO.

The airdrop and vesting contracts are based on the Merkledistributor patterns. Users can claim tokens by proving to be part of the merkleroot. It should be noted that the merkle proof does not enforce that the proof item is a leaf node or if the item is at the specific index. An attack on this is rather unlikely as it still means that someone would need to find a keccak preimage. Tokens that are not

claimed remain in the contract forever. Users have to trust the deployer of the vesting or airdrop contracts to provide sufficient tokens in order for everyone to be able to claim their share. There is no way to verify that there are enough tokens which might suggest that users may want to claim early. Some vestings can be revoked. However, they can only be revoked if they're claimed from the distributor. This means, that users might employ a strategy to claim late in order to avoid getting their vesting revoked. However, the deployer can claim + revoke for them (spending extra gas) to force a revocation.

5 Recommendations

5.1 Ensure that implementations of upgradeable contracts are initialized

Description

It is recommended to check whether implementations/logic contracts used with the OZ upgradability pattern may be left uninitialized. While these logic contracts are typically not consumed directly (they are only delegated to) they may still be claimable by anyone as the `initialize` function is not access protected. This is usually not a problem unless there's a way to self-destruct the contract. However, there is a risk of reputational damage if someone initialized the implementation in an attempt to carry out a malicious campaign potentially tricky users into believing this is the legitimate contract while it's only the logic contract for an upgradeable contract.

5.2 zDAO Token - reject calls that have no effect - zero value transfers

Description

Consider returning or bailing early for calls that have no effect on the system, like if the total amount transferred is zero (empty recipients, zero amount). Consider rejecting transfers to the contract address to avoid tokens getting stuck.

AO-Token/contracts/ZeroDAOToken.sol:L68-L78

```
function transferBulk(address[] calldata recipients, uint256 amount)
  external
  returns (bool)
{
  address sender = _msgSender();

  uint256 total = amount * recipients.length;
  require(
    _balances[sender] >= total,
    "ERC20: transfer amount exceeds balance"
  );
```

5.3 zDAO Token - check contract state before wasting gas on calculations

Description

Consider checking if a contract is paused as the first thing in the function to avoid unnecessarily wasting gas on calculations for a call that will always fail (if the contract is paused).

zDAO-Token/contracts/ZeroDAOToken.sol:L80-L81

```
require(!paused(), "ERC20Pausable: token transfer while paused");
```

6 Findings

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 zDAO Token - Specification violation - Snapshots are never taken Major Partially Addressed

Resolution

Addressed with zer0-os/zDAO-Token@ `81946d4` by exposing the `_snapshot()` method to a dedicated snapshot role (likely to be a DAO) and the owner of the contract.

We would like to note that we informed the client that depending on how the snapshot method is used and how predictably snapshots are consumed this might open up a frontrunning vector where someone observing that a `_snapshot()` is about to be taken might sandwich the snapshot call, accumulate a lot of stake (via 2nd markets, lending platforms), and returning it right after it's been taken. The risk of losing funds may be rather low (especially if performed by a miner) and the benefit from a DAO proposal using this snapshot might outweigh it. It is still recommended to increase the number of snapshots taken or take them on a regular basis (e.g. with every first transaction to the contract in a block) to make it harder to sandwich the snapshot taking.

Description

According to the zDAO Token specification the DAO token should implement a snapshot functionality to allow it being used for DAO governance votings.

> Any transfer, mint, or burn operation should result in a snapshot of the token balances of involved users being taken.

While the corresponding functionality is implemented and appears to update balances for snapshots, `_snapshot()` is never called, therefore, the snapshot is never taken. e.g. attempting to call `balanceOfAt` always results in an error as no snapshot is available.

zDAO-Token/contracts/ZeroDAOToken.sol:L12-L17

```
contract ZeroDAOToken is
  OwnableUpgradeable,
  ERC20Upgradeable,
  ERC20PausableUpgradeable,
  ERC20SnapshotUpgradeable
{
```

zDAO-Token/contracts/ZeroDAOToken.sol:L83-L83

```
_updateAccountSnapshot(sender);
```

Note that this is an explicit requirement as per specification but unit tests do not seem to attempt calls to `balanceOfAt` at all.

Recommendation

Actually, take a snapshot by calling `_snapshot()` once per block when executing the first transaction in a new block. Follow the openzeppeling documentation for ERC20Snapshot.

6.2 zDAO-Token - Revoking vesting tokens right before cliff period expiration might be delayed/front-runned Minor

Description

The owner of `TokenVesting` contract has the right to revoke the vesting of tokens for any `beneficiary`. By doing so, the amount of tokens that are already vested and weren't released yet are being transferred to the `beneficiary`, and the rest are being transferred to the owner. The beneficiary is expected to receive zero tokens in case the revocation transaction was executed before the cliff period is over. Although unlikely, the beneficiary may front run this revocation transaction

by delaying the revocation (and) or inserting a release transaction right before that, thus withdrawing the vested amount.

zDAO-Token/contracts/TokenVesting.sol:L69-L109

```
function release(address beneficiary) public {
  uint256 unreleased = getReleasableAmount(beneficiary);
  require(unreleased > 0, "Nothing to release");

  TokenAward storage award = getTokenAwardStorage(beneficiary);
  award.released += unreleased;

  targetToken.safeTransfer(beneficiary, unreleased);

  emit Released(beneficiary, unreleased);
}

/**
 * @notice Allows the owner to revoke the vesting. Tokens already vested
 * are transfered to the beneficiary, the rest are returned to the owner.
 * @param beneficiary Who the tokens are being released to
 */
function revoke(address beneficiary) public onlyOwner {
  TokenAward storage award = getTokenAwardStorage(beneficiary);

  require(award.revocable, "Cannot be revoked");
  require(!award.revoked, "Already revoked");

  // Figure out how many tokens were owed up until revocation
  uint256 unreleased = getReleasableAmount(beneficiary);
  award.released += unreleased;

  uint256 refund = award.amount - award.released;

  // Mark award as revoked
  award.revoked = true;
  award.amount = award.released;

  // Transfer owed vested tokens to beneficiary
  targetToken.safeTransfer(beneficiary, unreleased);
  // Transfer unvested tokens to owner (revoked amount)
  targetToken.safeTransfer(owner(), refund);

  emit Released(beneficiary, unreleased);
  emit Revoked(beneficiary, refund);
}
```

commendation

The issue described above is possible, but very unlikely. However, the `TokenVesting` owner should be aware of that, and make sure not to revoke vested tokens closely to cliff period ending.

6.3 zDAO-Token - Vested tokens revocation depends on claiming state Minor

Description

The owner of the `TokenVesting` contract can revoke the vesting of tokens for any beneficiary by calling `TokenVesting.revoke` only for tokens that have already been claimed using `MerkleTokenVesting.claimAward`. Although anyone can call `MerkleTokenVesting.claimAward` for a given beneficiary, in practice it is mostly the beneficiary's responsibility. This design decision, however, incentivizes the beneficiary to delay the call to `MerkleTokenVesting.claimAward` up to the point when he wishes to cash out, to avoid potential revocation. To revoke vesting tokens the owner will have to claim the award on the beneficiary's behalf first (which might be a gas burden), then call `TokenVesting.revoke`.

Examples

zDAO-Token/contracts/TokenVesting.sol:L86-L109

```
function revoke(address beneficiary) public onlyOwner {
  TokenAward storage award = getTokenAwardStorage(beneficiary);

  require(award.revocable, "Cannot be revoked");
  require(!award.revoked, "Already revoked");

  // Figure out how many tokens were owed up until revocation
  uint256 unreleased = getReleasableAmount(beneficiary);
  award.released += unreleased;

  uint256 refund = award.amount - award.released;

  // Mark award as revoked
  award.revoked = true;
  award.amount = award.released;

  // Transfer owed vested tokens to beneficiary
  targetToken.safeTransfer(beneficiary, unreleased);
  // Transfer unvested tokens to owner (revoked amount)
  targetToken.safeTransfer(owner(), refund);

  emit Released(beneficiary, unreleased);
  emit Revoked(beneficiary, refund);
}
```

Recommendation

Make sure that the potential owner of a `TokenVesting` contract is aware of this potential issue, and has the required processes in place to handle it.

6.4 zDAO-Token - Total amount of claimable tokens is not verifiable Minor ✓ Fixed

Description

Since both `MerkleTokenVesting` and `MerkleTokenAirdrop` use an off-chain Merkle tree to store the accounts that can claim tokens from the underlying contract, there is no way for a user to verify whether the contract token balance is sufficient for all claimers.

Recommendation

Make sure that users are aware of this trust assumption.

7 Document Change Log

Version	Date	Description
1.0	2021-05-20	Initial report
1.1	2021-08-23	Update: added section 3 - `WILD` Token

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 code and only the code we note as being within

the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as 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 specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

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.

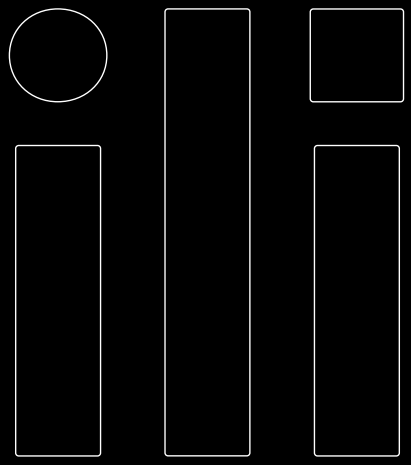
LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. 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.

Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

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*

liz.daldalian@gmail.com

→

POWERED BY CONSENSYS