Audit of iearn.finance by CryptoManiacs

Authors:

- Mikhail Melnik
- Anton Bukov

Scope:

- https://github.com/iearn-finance/itoken/blob/eb08d0b7db86f532620c8f98b3cea10847b329fd/contracts/YDAI.sol (https://github.com/iearn-finance/itoken/blob/eb08d0b7db86f532620c8f98b3cea10847b329fd/contracts/YDAI.sol)

Summary:

Provided smart contracts present lending system aggregator, automatically switching among lending opportunities of Compound, DyDx, Fulcrum, Aave platforms.

CryptoManiacs team performed security audit of iearn.finance (https://iearn.finance) smart contract including automatic and manual source code analysis. The issues we found were classified by severity (see Appendix 1) and included in the following report.

1. Critical Issues

1.1. `set_new_*` methods allow owner to drain all the funds

- *Severity*: Critical

- *Location*: `YDAI.sol#L404-L424` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L404-L424)

- *Comment*:

  These methods allow Owner to change addresses of Compound, DyDx, Fulcrum, Aave platforms. They can be utilized to drain all the funds by upgrading to the malicious contract that forwards all the funds to the attacker.

- *Status*:
    - *Developers*: Removed in v2, commit 4a2b9cf4e29a6d057f468be56e3205b1ec9d18e9 (https://github.com/iearn-finance/itoken/commit/4a2b9cf4e29a6d057f468be56e3205b1ec9d18e9)
    - *Auditors*: verified (Anton)

1.2 `inCaseTokenGetsStuck` allows Owner to drain all the funds

- *Severity*: Critical

- *Location*: `YDAI.sol#L717-L721` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L717-L726)

- *Comment*:

  `inCaseTokenGetsStuck` allows Owner to drain all the funds from the contract.

- *Status*:
    - *Developers*: Removed in v2, commit 4a2b9cf4e29a6d057f468be56e3205b1ec9d18e9 (https://github.com/iearn-finance/itoken/commit/4a2b9cf4e29a6d057f468be56e3205b1ec9d18e9)
    - *Auditors*: verified (Anton)

2. Major Issues

2.1. Missing `withdrawSomeFulcrum()` method.

- *Severity*: High

- *Location*: `YDAI.sol#L586` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L586)

- *Comment*:
    - Missing `withdrawSomeFulcrum()` method similar to `withdrawSomeCompound()` with scaling amount from underlying asset.

- *Status*:
    - *Developers*: Added in v2, commits 8d6b635355174110a41401635d3b2b74958ca2b8 (https://github.com/iearn-finance/itoken/commit/8d6b635355174110a41401635d3b2b74958ca2b8),

b6a2bec292288ccb3369d4410963e44cb32e7e85 (https://github.com/iearn-finance/itoken/commit/b6a2bec292288ccb3369d4410963e44cb32e7e85)
    - *Auditors*: verified (Anton)

3. Medium Issues

3.1. Unnecessary ERC20 library modification

- *Severity*: Medium

- *Location*:

    - `YDAI.sol#L699-L702` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L699-L702)
    - `YDAI.sol#L691` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L691)

- *Comment*:

    - We noticed you modified OpenZeppelin's ERC20 implementation to have unsafe direct access to `_balances` and `_totalSupply` members from inherited smart contract. You can achieve the same behaviour by calling `_burn(msg.sender, _shares)` method. OpenZeppelin's ERC20 implementation eposes internal API and protects private members to prevent misuse. **Any changes in external libraries must be documented.**
    - Second link points to duplicate check, this issue is handled by SafeMath in your code below and in `_burn()` method.

- *Status*:

    - *Developers*: Left unmodified for now.
    - *Developers*: All the changes to original libraries were reverted.
    - *Auditors*: verified (Mikhail, Anton)

3.2. Wrong calculation for partial withdrawal

- *Severity*: Medium

- *Location*: `YDAI.sol#L707` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L707)

- *Comment*:

    - Since we already have some amount `b`, we can withdraw only balance minus redeem amount: `withdrawSome(r.sub(b))` Moreover `withdrawSome(r)` will revert if significant part of the DAI is stored directly on the contract so there would be not enough DAI to withdraw from lending provider.
      Also if the following call to `rebalance()` will actually do the rebalancing then it might be cheaper in terms of gas usage to `withdrawAll()` instead of `withdrawSome()`.

- *Status*:

    - *Developers*: Updated in v2, commit daec3a0131d1a09761cd9573722060884579cca5 (https://github.com/iearn-finance/itoken/commit/daec3a0131d1a09761cd9573722060884579cca5)

    - *Auditors*: verified (Anton)

3.3. Continious jumps among pools in case of huge pool size and APR slippage

- *Severity*: Medium

- *Location*: `YDAI.sol` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol)

- *Comment*:

    - When size of the pool will became significant ($0.5M-$1M on current market) jumping between protocols would affect their APR significantly. Increasing pool size for 10% leads for APR drop for 1/10.

- *Status*:

    - *Developers*: -

4. Minor Issues

4.1. Struct types access and initialization

- *Severity*: Low

- *Location*:

    - `YDAI.sol#L368` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L368)

    - `YDAI.sol#L466` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L466)
    - `YDAI.sol#L484` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L484)

- *Comment*:
    - No need to inherit from `Structs`, you can use structs from different smart contract like this: `Structs.AssetAmount`.

    - Instantiation of struct better to perform with following syntax for `AssetAmount`, `ActionArgs` and other structures too:

```
Structs.AssetAmount memory amt = Structs.AssetAmount({
    sign: true,
    denomination: AssetDenomination.Wei,
    ref: AssetReference.Delta,
    value: amount
});
```

- *Status*:
    - *Developers*: -

4.2. Excess infinite approve

- *Severity*: Low

- *Location*: `YDAI.sol#L512` (https://github.com/iearn-finance/itoken/blob/master/contracts/YDAI.sol#L512)

- *Comment*:
    - No need to create infinite approve to **Aave lending pool**, approve to **Aave lending pool core** is enough.
- *Status*:
    - *Developers*: Removed in v2, commit 5f25ad1c3f911abe0dcc31d8a8d1556623a83cbe (https://github.com/iearn-finance/itoken/commit/5f25ad1c3f911abe0dcc31d8a8d1556623a83cbe)
    - *Auditors*: verified (Anton)

5. Note issues

5.1. Consider to add prefix undescore to internal methods

- *Severity*: Note

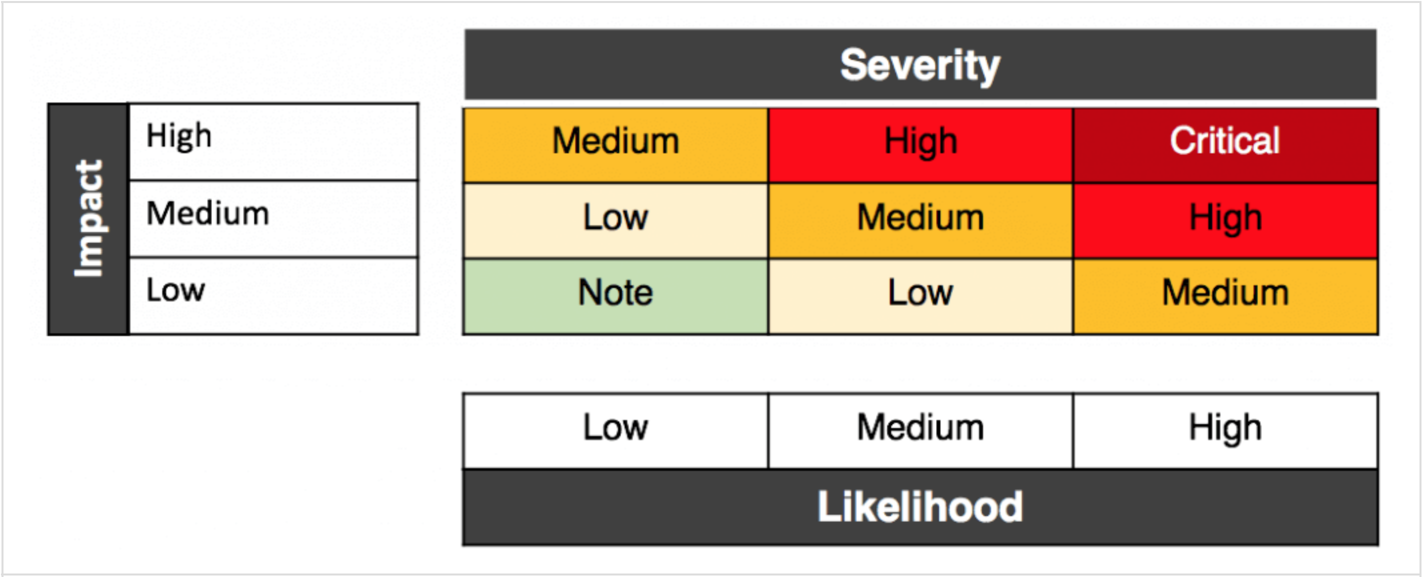
- *Location*: Whole file

- *Status*:
  - *Developers*: Updated in v2, commit [2e2c3b1342c3bb8e124f2e7a218b4957e35cfa79](https://github.com/iearn-finance/itoken/commit/2e2c3b1342c3bb8e124f2e7a218b4957e35cfa79) (https://github.com/iearn-finance/itoken/commit/2e2c3b1342c3bb8e124f2e7a218b4957e35cfa79)
  - *Auditors*: verified (Anton)

Appendix 1 - Terminology

Severity

Assessment of the magnitude of an issue.


Severity
Impact
High
Medium
Low
Medium
High
Critical
Low
Medium
High
Note
Low
Medium
Low
Medium
High
Likelihood

Picture 1. Severity

Minor (Low)

Minor issues are generally subjective in nature or potentially associated with the topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code. The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

Medium

Medium issues are generally objective in nature but do not represent any actual bugs or security problems.

These issues should be addressed unless there is a clear reason not to.

Major (High)

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise in order to be exploited.

If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

Critical

Critical issues are directly exploitable bugs or security vulnerabilities.

If unaddressed, these issues are likely or guaranteed to cause major problems or ultimately a full failure in the operations of the contract.