# DYP TOKEN AUDIT

**October 2020**
**Revised on January 2021 (Version 1.1)**

## BLOCKCHAIN CONSILIUM

# Contents

# Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND BLOCKCHAIN CONSILIUM DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH BLOCKCHAIN CONSILIUM.

## Purpose of the report

The Audits 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 the Solidity programming language that could present security risks. Cryptographic tokens and smart contracts are emergent technologies and carry with them high levels of technical risk and uncertainty.

The Audits are not an endorsement or indictment of any particular project or team, and the Audits 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 Audits in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. 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. There is no owed duty to any Third-Party by virtue of publishing these Audits.

# Introduction

We first thank dyp.finance for giving us the opportunity to audit their smart contract. This document outlines our methodology, audit details, and results.

dyp.finance asked us to review their DYP Token smart contract (ETH mainnet address: `0x961C8c0B1aaD0c0b10a51FeF6a867E3091BCef17`). Blockchain Consilium reviewed the system from a technical perspective looking for bugs, issues and vulnerabilities in their code base. The Audit is valid for `0x961C8c0B1aaD0c0b10a51FeF6a867E3091BCef17` Ethereum Smart Contract only. The audit is not valid for any other versions of the smart contract. Read more below.

# Audit Summary

This code is clean, thoughtfully written and in general well architected. The code conforms closely to the documentation and specification.

The code is based on OpenZeppelin in many cases. In general, OpenZeppelin's codebase is good, and this is a relatively safe start.

Overall, the code is well commented and clear on what it is supposed to do for each function. The visibility and state mutability of all the functions are clearly specified, and there are no confusions.

| **Audit Result** | **PASSED** |
|---|---|
| High Severity Issues | None |
| Moderate Severity Issues | None |
| Low Severity Issues | 1 |
| Informational Observations | 1 |

# Overview

The project has one Solidity file for the DYP ERC20 Token Smart Contract, the DeFiYieldProtocol.sol file that contains about 1032 lines of Solidity code. We manually reviewed each line of code in the smart contract. All the functions and state variables are well commented using the NatSpec documentation for the functions which is good to understand quickly how everything is supposed to work.

*Nice Features:*
The contract provides a good suite of functionality that will be useful for the entire contract AND It **USES** SafeMath library to check for overflows and underflows, which protects against overflow and underflow attacks. All the ERC20 functions are included; it is a valid ERC20 token and in addition has some extra functionality mitigating approval-doublespend attacks and governance integration which is based on Compund like governance or $CORE token like governance.

### Methodology:

Blockchain Consilium manually reviewed the smart contract line-by-line, keeping in mind industry best practices and known attacks, looking for any potential issues and vulnerabilities, and areas where improvements are possible.

We also used automated tools like slither for analysis and reviewing the smart contract. The raw output of these tools is included in the Appendix. These tools often give false-positives, and any issues reported by them but not included in the issue list can be considered not valid.

### Classification / Issue Types Definition:

1. **High Severity:** which presents a significant security vulnerability or failure of the contract across a range of scenarios, or which may result in loss of funds.
2. **Moderate Severity:** which affects the desired outcome of the contract execution or introduces a weakness that can be exploited. It may not result in loss of funds but breaks the functionality or produces unexpected behaviour.
3. **Low Severity:** which does not have a material impact on the contract execution and is likely to be subjective.

The smart contract is considered to pass the audit, as of the audit date, if no high severity or moderate severity issues are found.

## Attacks & Issues considered while auditing

In order to check for the security of the contract, we reviewed each line of code in the smart contract considering several known Smart Contract Attacks & known issues.

- **Overflows and underflows:**
  An overflow happens when the limit of the type variable uint256 , 2 ** 256, is

exceeded. What happens is that the value resets to zero instead of incrementing more.

For instance, if we want to assign a value to a uint bigger than 2 ** 256 it will simple go to 0—this is dangerous.

On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0.For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1.

This is quite dangerous. This contract **DOES** check for overflows and underflows, using **OpenZeppelin's** *SafeMath* for overflow and underflow protection.

- **Reentrancy Attack:**
  One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. This class of bug can take many forms, and both of the major bugs that led to the DAO's collapse were bugs of this sort.

  This smart contract follows *checks-effects-interactions* pattern and thus *is not found vulnerable* to re-entrancy attack.

- **Replay attack:**
  The replay attack consists of making a transaction on one blockchain like the original Ethereum's blockchain and then repeating it on another blockchain like the Ethereum's classic blockchain. The ether is transferred like a normal transaction from a blockchain to another. Though it's no longer a problem because since the version 1.5.3 of *Geth* and 1.4.4 of *Parity* both implement the attack protection EIP 155 by Vitalik Buterin.
  So the people that will use the contract depend on their own ability to be updated with those programs to keep themselves secure.

- **Short address attack:**
  This attack affects ERC20 tokens, was discovered by the Golem team and consists of the following:

  A user creates an Ethereum wallet with a trailing 0, which is not hard because it's only a digit. For instance: `0xiofa8d97756as7df5sd8f75g8675ds8gsdg0`
  Then he buys tokens by removing the last zero:
  Buy 1000 tokens from account `0xiofa8d97756as7df5sd8f75g8675ds8gsdg.` If the contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeroes to the transaction until the address is complete.

The virtual machine will return 256000 for each 1000 tokens bought. This is abug of the virtual machine.

Here is a **fix for short address attacks**

```
modifier onlyPayloadSize(uint size) {
    assert(msg.data.length >= size + 4);
    _;
}
function transfer(address _to, uint256 _value) onlyPayloadSize(2 * 32) {
    // do stuff
}
```

*Whether or not it is appropriate for token contracts to mitigate the shortaddress attack is a contentious issue among smart-contract developers. Many, including those behind the OpenZeppelin project, have explicitly chosen not to do so. Blockchain Consilium doesn't consider short address attack an issue of the smart contract at the token smart contract level.*

This contract **does not** implement an onlyPayloadSize(uint numwords) modifier for transfer, transferFrom, approve, increaseApproval, and decreaseApproval functions, it probably assumes that checks for short address attacks are handled at a higher layer (which generally are), and since the onlyPayloadSize() modifier started causing some bugs restricting the flexibility of the smart contracts, it's alright not to check for short address attacks at the Token Contract level to allow for some more flexibility for dAPP coding, but the checks for short address attacks must be done at some layer of coding (e.g. for buys and sells, the exchange can do it - almost all wellknown exchanges check for short address attacks after it was found), this contract *does not prevent short address attack, so the checks for short address attack must be done while buying or selling or coding a DAPP using DYP where necessary.*

You can read more about the attack here: ERC20 Short Address Attacks.

- **Approval Double-spend**

  ERC20 Standard allows users to approve other users to manage their tokens, or spend tokens from their account till a certain amount, by setting the user's allowance with the standard `approve` function, then the allowed user may use `transferFrom` to spend the allowed tokens.

  Hypothetically, given a situation where Alice approves Bob to spend 100 Tokens from her account, and if Alice needs to adjust the allowance to allow Bob to spend 20 more tokens, normally – she'd check Bob's allowance (100 currently) and start a new `approve` transaction allowing Bob to spend a total of 120 Tokens instead of 100 Tokens.

Now, if Bob is monitoring the Transaction pool, and as soon as he observes new transaction from Alice approving more amount, he may send a `transferFrom` transaction spending 100 Tokens from Alice's account with higher gas price and do all the required effort to get his spend transaction mined before Alice's new approve transaction.

Now Bob has already spent 100 Tokens, and given Alice's approve transaction is mined, Bob's allowance is set to 120 Tokens, this would allow Bob to spend a total of 100 + 120 = 220 Tokens from Alice's account instead of the allowed 120 Tokens. This exploit situation is known as Approval Double-Spend Attack.

A potential solution to minimize these instances would be to set the non-zero allowance to 0 before setting it to any other amount.

It's possible for `approve` to enforce this behaviour without interface changes in the ERC20 specification:

```
if ((_value != 0) && (approved[msg.sender][_spender] != 0)) return false;
```

However, this is just an attempt to modify user behaviour. If the user does attempt to change from one non-zero value to another, the double spend might still happen, since the attacker may set the value to zero by already spending all the previously allowed value before the user's new approval transaction.

If desired, a non-standard function can be added to minimize hassle for users. The issue can be fixed with minimal inconvenience by taking a change value rather than a replacement value:

```
function increaseAllowance (address _spender, uint256 _addedValue)
returns (bool success) {
  uint oldValue = approved[msg.sender][_spender];
  approved[msg.sender][_spender] = safeAdd(oldValue, _addedValue);
  return true;
}
```

Even if this function is added, it's important to keep the original for compatibility with the ERC20 specification.

This contract implements an `increaseApproval` and a `decreaseApproval` function, both of which takes the change in value instead of taking the new value, which is really *nice*.

For more, see this discussion on GitHub:
https://github.com/ethereum/EIPs/issues/20#issuecomment263524729

- ## Accidental Token Loss

    - Token Smart Contracts should prevent transferring tokens to the token smart contract address if there's no good reason to not prevent, or if there's no way to take out tokens held by the token smart contract. The DYP smart contract *does not* prevent transferring of DYP to DYP smart contract address. If someone accidentally sends DYP to the DYP smart contract, the tokens may be irrevocably locked. If desired, a nonstandard function may be added to allow owner to transfer out token balance from this smart contract, or the _transfer function may be edited to disallow transfers to `address(this)`. This is subjective and depends on project's plan whether they want to implement it or not.

    - One more issue is when other ERC20 Tokens are transferred to the DYP smart contract, traditionally there would've been no way to take them out, if desired, this may be solved by implementing a function to allow owner to transfer out any ERC20 tokens from this smart contract. This is subjective and depends on project's plan whether they want to implement it or not.

- ## Sybil Attack

    The vote delegation functionality is vulnerable to sybil attack, for example a token holder can `_delegate` their voting power to an Ethereum account and move their tokens to another account, and `_delegate` again from new account because the token transfer methods do not move delegates along with token transfer – amplifying the user's voting power.

    Fix: _moveDelegates in transfer and transferFrom.

    Response: The DYP team has deployed another governance smart contract and decided to use this smart contract for ERC20 Token functionality only which is not impacted by this issue.

# Issues Found & Informational Observations

## High Severity Issues

No high severity issues were found in the smart contract.

## Moderate Severity Issues

No moderate severity issues were found in the smart contract.

## Low Severity Issues

- **Sybil Attack:**

  The vote delegation functionality is vulnerable to sybil attack, for example a token holder can `_delegate` their voting power to an Ethereum account and move their tokens to another account, and `_delegate` again from new account because the token transfer methods do not move delegates along with token transfer – amplifying the user's voting power.

  Fix: _moveDelegates in transfer and transferFrom.

  Response: The DYP team has deployed another governance smart contract and decided to use this smart contract for ERC20 Token functionality only which is not impacted by this issue.

## Informational Observations

If someone accidentally sends DYP token to DYP token smart contract, those tokens will be potentially irrevocably locked in this smart contract, similarly any other tokens sent to this smart contract might be irrevocably locked.

This is subjective to the project and depends on their plan, whether they want to allow funds recovery (ERC20 compliant token recovery) by admin / owner from this contract or not.

If desired, a non-standard function may be added to allow owner to recover tokens transferred to `address(this)`, and DYP transfers to `address(this)` may be blocked to prevent these type of accidental future events.

# Line by line comments

- Line 9:
  The compiler version is specified as ^0.6.0, this means the code can be compiled with solidity compilers with versions equal to or greater than 0.6.0 only, that's a best practise. The latest compiler version at the time of auditing is 0.7.3.

- Lines 11 to 30:
  Context helper function is included to provide information about transaction sender and transaction data. This function may be useful for implementing GSN compatible contracts.

- Lines 38 to 110:
  IERC20 standard interface is included with appropriate function signatures for standard ERC20 functions.

- Lines 118 to 272:
  SafeMath library is included for safe arithmetic operations.

- Lines 280 to 416:
  Address library is included to provide helper address functions to check whether an address is a smart contract or not, to transfer ether to addresses, and several other helper functions for performing solidity calls.

- Lines 428 to 725:
  Standard ERC20 implementation is included, inheriting from Context, and IERC20. This also implements increaseAllowance and decreaseAllowance nonstandard functions for mitigation of approval doublespend attack.

- Lines 733 to 795:
  The `Ownable` contract makes the contract creator the owner of the contract, so that in `DYP` the contract creator becomes the owner and receives the initially minted tokens, the `Ownable` contract has following noteworthy functions:

    - `transferOwnership` allows the current owner transfer the ownership of the contract to a new Ethereum address when needed.
    - `renounceOwnership` allows the current owner to renounce their ownership and make the contract unowned. Renouncing ownership will disable onlyOwner functions of the smart contract, currently

onlyOwner functions include transferOwnership and renounceOwnership.

- Lines 803 to 1032:
  The DYP contract is implemented, assigning the creator as owner and sending 30,000,000 DYP with 18 decimals to owner. It also implements a non-standard functions for governance / votes integration which is based on $CORE style governance (which in turn is based on $COMP style governance) which provides functionality for delegating votes and calculating checkpoints for marking number of votes from given blocks.

  NOTE: The vote delegation functionality is vulnerable to sybil attacks that can cause vote amplification, this smart contract is not used for voting or governance functionality, a different governance smart contract is deployed for voting, thus the token contact is considered safe from this issue.

# Appendix

## Smart Contract Summary

- Contract Context
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)
- Contract IERC20
  - From IERC20
    - allowance(address,address) (external)
    - approve(address,uint256) (external)
    - balanceOf(address) (external)
    - totalSupply() (external)
    - transfer(address,uint256) (external)
    - transferFrom(address,address,uint256) (external)
- Contract SafeMath (Most derived contract)
  - From SafeMath
    - add(uint256,uint256) (internal)
    - div(uint256,uint256) (internal)
    - div(uint256,uint256,string) (internal)
    - mod(uint256,uint256) (internal)
    - mod(uint256,uint256,string) (internal)
    - mul(uint256,uint256) (internal)
    - sub(uint256,uint256) (internal)
    - sub(uint256,uint256,string) (internal)
- Contract Address (Most derived contract)
  - From Address
    - _functionCallWithValue(address,bytes,uint256,string) (private)
    - functionCall(address,bytes) (internal)
    - functionCall(address,bytes,string) (internal)
    - functionCallWithValue(address,bytes,uint256) (internal)
    - isContract(address) (internal)
    - sendValue(address,uint256) (internal)
- Contract ERC20
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal) ○ From ERC20
    - _approve(address,address,uint256) (internal)
    - _beforeTokenTransfer(address,address,uint256) (internal)
    - _burn(address,uint256) (internal)

        - _mint(address,uint256) (internal)
        - _setupDecimals(uint8) (internal)
        - _transfer(address,address,uint256) (internal)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - constructor(string,string) (public)
        - decimals() (public)
        - decreaseAllowance(address,uint256) (public)
        - increaseAllowance(address,uint256) (public)
        - name() (public)
        - symbol() (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)
- Contract Ownable
    - From Context
        - _msgData() (internal)
        - _msgSender() (internal) ○ From Ownable
        - constructor() (internal)
        - owner() (public)
        - renounceOwnership() (public)
        - transferOwnership(address) (public)
- Contract DeFiYieldProtocol (Most derived contract)
    - From Ownable
        - owner() (public)
        - renounceOwnership() (public)
        - transferOwnership(address) (public) ○ From Context
        - _msgData() (internal)
        - _msgSender() (internal) ○ From ERC20
        - _approve(address,address,uint256) (internal)
        - _beforeTokenTransfer(address,address,uint256) (internal)
        - _burn(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _setupDecimals(uint8) (internal)
        - _transfer(address,address,uint256) (internal)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - constructor(string,string) (public)
        - decimals() (public)
        - decreaseAllowance(address,uint256) (public)

- increaseAllowance(address,uint256) (public)
- name() (public)
- symbol() (public)
- totalSupply() (public)
- transfer(address,uint256) (public)
- transferFrom(address,address,uint256) (public) ○ From DeFiYieldProtocol
- _delegate(address,address) (internal)
- _moveDelegates(address,address,uint256) (internal)
- _writeCheckpoint(address,uint32,uint256,uint256) (internal)
- constructor() (public)
- delegate(address) (external)
- delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (external)
- delegates(address) (external)
- getChainId() (internal)
- getCurrentVotes(address) (external)
- getPriorVotes(address,uint256) (external)
- safe32(uint256,string) (internal)

## Slither Results

```
> slither defiyieldprotocol.sol

Compilation warnings/errors on defiyieldprotocol.sol:
Warning: SPDX license identifier not provided in source file. Before publishing, consider
adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file.
Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see
https://spdx.org for more information. --> defiyieldprotocol.sol

Warning: Documentation tag on non-public state variables will be disallowed in 0.7.0. You
will need to use the @dev tag explicitly.
   --> defiyieldprotocol.sol:809:5:
    |
809 |     /// @notice A record of each accounts delegate     |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

INFO:Detectors:
DeFiYieldProtocol._writeCheckpoint(address,uint32,uint256,uint256)
(defiyieldprotocol.sol#1002-1020) uses a dangerous strict equality:
-      nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock
== blockNumber (defiyieldprotocol.sol#1012)
Reference:
https://github.com/crytic/slither/wiki/DetectorDocumentation#dangerous-
strict-equalities INFO:Detectors:
ERC20.constructor(string,string).name (defiyieldprotocol.sol#475) shadows:
-      ERC20.name() (defiyieldprotocol.sol#484-486) (function)
ERC20.constructor(string,string).symbol (defiyieldprotocol.sol#475) shadows:
-      ERC20.symbol() (defiyieldprotocol.sol#492-494) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#localvariable-
shadowing INFO:Detectors:
DeFiYieldProtocol.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)
(defiyieldprotocol.sol#868-909) uses timestamp for comparisons    Dangerous
comparisons:
-      require(bool,string)(now <= expiry,DYP::delegateBySig: signature expired)
(defiyieldprotocol.sol#907)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#blocktimestamp
INFO:Detectors:
Address.isContract(address) (defiyieldprotocol.sol#301-310) uses assembly
-      INLINE ASM (defiyieldprotocol.sol#308)
Address._functionCallWithValue(address,bytes,uint256,string)
(defiyieldprotocol.sol#394-415) uses assembly
-      INLINE ASM (defiyieldprotocol.sol#407-410)
DeFiYieldProtocol.getChainId() (defiyieldprotocol.sol#1027-1031) uses assembly       -
INLINE ASM (defiyieldprotocol.sol#1029)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assemblyusage
INFO:Detectors:
Different versions of Solidity is used in :
-      Version used: ['0.6.12', '^0.6.0', '^0.6.2']
-      ^0.6.0 (defiyieldprotocol.sol#9)
-      ^0.6.0 (defiyieldprotocol.sol#36)
-      ^0.6.0 (defiyieldprotocol.sol#116)
-      ^0.6.2 (defiyieldprotocol.sol#278)
-      ^0.6.0 (defiyieldprotocol.sol#422)
-      ^0.6.0 (defiyieldprotocol.sol#731)      - 0.6.12 (defiyieldprotocol.sol#799)
Reference:
https://github.com/crytic/slither/wiki/DetectorDocumentation#different-
pragma-directives-are-used INFO:Detectors:
Pragma version^0.6.0 (defiyieldprotocol.sol#9) allows old versions
Pragma version^0.6.0 (defiyieldprotocol.sol#36) allows old versions
Pragma version^0.6.0 (defiyieldprotocol.sol#116) allows old versions
Pragma version^0.6.2 (defiyieldprotocol.sol#278) allows old versions
Pragma version^0.6.0 (defiyieldprotocol.sol#422) allows old versions
Pragma version^0.6.0 (defiyieldprotocol.sol#731) allows old versions
Pragma version0.6.12 (defiyieldprotocol.sol#799) necessitates a version too recent to
be trusted. Consider deploying with 0.6.11 solc-0.6.12 is not recommended for
deployment
Reference: https://github.com/crytic/slither/wiki/DetectorDocumentation#incorrect-
versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (defiyieldprotocol.sol#328334):
-      (success) = recipient.call{value: amount}() (defiyieldprotocol.sol#332) Low
level call in Address._functionCallWithValue(address,bytes,uint256,string)
(defiyieldprotocol.sol#394-415):
-      (success,returndata) = target.call{value: weiValue}(data)
(defiyieldprotocol.sol#398)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#lowlevel-
calls INFO:Detectors:
Variable DeFiYieldProtocol._delegates (defiyieldprotocol.sol#810) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/DetectorDocumentation#conformity-to-
solidity-naming-conventions
INFO:Detectors:
DeFiYieldProtocol.constructor() (defiyieldprotocol.sol#805-807) uses literals with too
many digits:
-      _mint(msg.sender,300000000000000000000000000)
(defiyieldprotocol.sol#806)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-manydigits
INFO:Detectors:
symbol() should be declared external:
-      ERC20.symbol() (defiyieldprotocol.sol#492-494) decimals() should be declared
external:
-      ERC20.decimals() (defiyieldprotocol.sol#509-511) totalSupply() should be
declared external:
-      ERC20.totalSupply() (defiyieldprotocol.sol#516-518) transfer(address,uint256)
should be declared external:
-      ERC20.transfer(address,uint256) (defiyieldprotocol.sol#535-538)
allowance(address,address) should be declared external:
-      ERC20.allowance(address,address) (defiyieldprotocol.sol#543-545)
```

```
approve(address,uint256) should be declared external:
-      ERC20.approve(address,uint256) (defiyieldprotocol.sol#554-557)
transferFrom(address,address,uint256) should be declared external:
-      ERC20.transferFrom(address,address,uint256) (defiyieldprotocol.sol#571-
575) increaseAllowance(address,uint256) should be declared external:
         - ERC20.increaseAllowance(address,uint256) (defiyieldprotocol.sol#589-
592) decreaseAllowance(address,uint256) should be declared external:
         - ERC20.decreaseAllowance(address,uint256) (defiyieldprotocol.sol#608-
611) owner() should be declared external:
-      Ownable.owner() (defiyieldprotocol.sol#762-764) renounceOwnership() should be
declared external:
-      Ownable.renounceOwnership() (defiyieldprotocol.sol#781-784)
transferOwnership(address) should be declared external:
-      Ownable.transferOwnership(address) (defiyieldprotocol.sol#790-794) Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#publicfunction-that-
could-be-declared-external
INFO:Slither:defiyieldprotocol.sol analyzed (7 contracts with 46 detectors), 32
result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github
integration
```