CERTIK

Security Assessment

DYP Finance

Apr 23rd, 2021

Summary

This report has been prepared for DYP Finance smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	DYP Finance
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/dypfinance/dyp-earn-vault/tree/aa3bce58a0331e44c5fa343013e4b677852e9f4d
Commits	1. aa3bce58a0331e44c5fa343013e4b677852e9f4d 2. 8bf1eea9f88be9d26bfc93311bdec0d503f1d405

Audit Summary

Delivery Date	Apr 23, 2021
Audit Methodology	Static Analysis, Manual Review, Testnet Deployment
Key Components	

Vulnerability Summary

Total Issues	12
● Critical	0
● Major	0
● Minor	6
● Informational	6
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
DIS	DISCLAIMER.txt	6d9e808298d10f197b39e51098103559b275382153cad9e3d94d5a249098cbba
LIC	LICENSE	765557294dbaf52e0551e97f095e7abb70200a18ccce22f4a14188ef87d5c3d5
LIE	LICENSE.txt	cde8537c66976881cdb4239dcc75beac5e7a0c64d04f8fa7ef7c5a54fd72b317
REA	README.md	fff5c8dfad3ee3be22213db9e5e23b65e4b5a30603a8718a93bc6e716b072280
VAU	vault-weth.sol	5a022f6b6139fd0033407acf825f028edb152b53c0f3e63b95edecb36c5c3737
VAL	vault.sol	672b1de51485897d6ded564abe9fd4f57a8529e3dd6ddb31657dfed876817fd2

Understandings

The following section demonstrates the purpose of each function in the Vault contract:

updateaccount()

The function first records the current dividend percentage of ETH, cToken, and platform token and saves it to lastxxxDivPoints[account]. If the user has deposited previously, calculate the reward he got since his last operation(deposit, withdraw or claim) and save it to the variable "xxxDivsBalance[account]."

deposit()

The function first called the "updateaccount" function to update the "lastxxxDivPoints[account]" and user reward balance. Then, deposit the amount of TRUSTED_DEPOSIT_TOKEN_ADDRESS tokens with minimum of 400k Gwei gas fees. Deposited tokens are forwarded to TRUSTED_CTOKEN_ADDRESS to mint cTokens. Increase the token balance of the user accordingly. 25% of 400k Gwei fees are used to buy back TRUSTED_PLATFORM_TOKEN_ADDRESS for burning. The remaining gas fees are added to totalEthDivPoints for dividends.

withdraw()

The function first called the "updateaccount" function to update the "lastxxxDivPoints[account]" and user reward balance. Withdraw amount of TRUSTED_DEPOSIT_TOKEN_ADDRESS tokens previously deposited with 400k Gwei gas fees. The amount must be less than the deposited amount and amount of TRUSTED_DEPOSIT_TOKEN_ADDRESS tokens exchanged by cTokens owned. The contract deducts 0.3% of the redeemed tokens as withdrawal fees and transfers the remaining token to the caller. The contract uses 25% of withdrawal fees to buy back TRUSTED_PLATFORM_TOKEN_ADDRESS tokens for burning and update the variable "totalTokenDivPoints" for dividends. 25% of 400k Gwei fees are used to buy back TRUSTED_PLATFORM_TOKEN_ADDRESS for burning. Then update the variable "totalEthDivPoints" for dividends.

claimEthDivs()

"claimEthDivs" claims all ETH dividends, from 75% of the 400kG ETH fees during deposit and withdrawal. The function first called the "updateaccount" function to update the "lastxxxDivPoints[account]" and user reward balance. The function then reduces the variable "ethDivsBalance[account]" to zero, reduces the ETH balance of the contract, and sends ETH reward to the user.

claimTokenDivs()

"claimTokenDivs" claims all TRUSTED_DEPOSIT_TOKEN_ADDRESS tokens which comes from 0.3%*(1-25%)*withdral_amount. The function first called the "updateaccount" function to update the "lastxxxDivPoints[account]" and user reward balance. The function then reduces the variable "tokenDivsBalance[account]" to zero, reduces the token balance of the contract, and sends deposited token rewards to the user.

claimCompoundDivs()

The function first called the "updateaccount" function to update the "lastxxxDivPoints[account]" and user reward balance. The function claims all interests earned from cTokens from the Compound pool and then redeems the underlying DEPOSIT_TOKENS without withdrawing fees.

claimPlatformTokenDivs()

The function first called the "updateaccount" function to update the "lastxxxDivPoints[account]" and user reward balance. The function claims all platform token rewards, which come from 2% of deposited tokens in the period of staking time. If there are not enough platform tokens to claim, the transaction will be reverted.

tokenDivsOwing()

"tokenDivsOwing" calculates the user's token dividends, which are currently not distributed.

ethDivsOwing()

"ethDivsOwing" calculates the user's ETH dividends which are currently not distributed.

platformTokenDivsOwing()

"platformTokenDivsOwing" calculates the user's platform token dividends which are currently not distributed.

getDepositorsList()

The function retrieves depositors who have TRUSTED_DEPOSIT_TOKEN_ADDRESS tokens in the contract.

Functions for the owner of the contract:

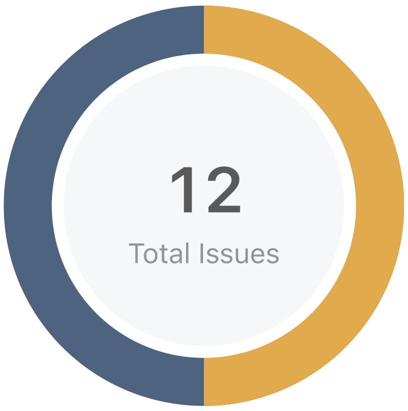
addPlatformTokenBalance()

The function adds platform tokens for the contract by transferring platform token from the contract owner to the contract.

claimExtraTokens()

The owner of the contract can call this function to transfer out tokens that are accidentally sent to this smart contract. The following section demonstrates the purpose of each function in the Vault contract:

Findings



🟥	Critical	**0** (0.00%)
🟧	Major	**0** (0.00%)
🟨	Minor	**6** (50.00%)
🟦	Informational	**6** (50.00%)
🟩	Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
VAL-01	Missmatched Approve amount	Inconsistency, Volatile Code	● Minor	✓ Resolved
VAL-02	function updateAccount called multi-time	Gas Optimization	● Informational	ⓘ Acknowledged
VAL-03	Check-Return first to save gas	Gas Optimization	● Informational	ⓘ Acknowledged
VAL-04	Return value not handled	Volatile Code	● Informational	ⓘ Acknowledged
VAL-05	3rd Party Dependencies	Control Flow	● Minor	ⓘ Acknowledged
VAL-06	Transaction fee not handled in function emergencyWithdraw	Logical Issue	● Minor	ⓘ Acknowledged
VAU-01	Missmatched Approve amount	Inconsistency, Volatile Code	● Minor	✓ Resolved
VAU-02	function updateAccount called multi-time	Gas Optimization	● Informational	ⓘ Acknowledged
VAU-03	Check-Return first to save gas	Gas Optimization	● Informational	ⓘ Acknowledged
VAU-04	Return value not handled	Volatile Code	● Informational	ⓘ Acknowledged
VAU-05	3rd Party Dependencies	Control Flow	● Minor	ⓘ Acknowledged
VAU-06	Transaction fee not handled in function emergencyWithdraw	Logical Issue	● Minor	ⓘ Acknowledged

VAL-01 | Missmatched Approve amount

Category	Severity	Location	Status
Inconsistency, Volatile Code	● Minor	vault.sol: 1258~1266(Vault)	✓ Resolved

Description

In function "handleFee," the contract called "safeApprove" allows Uniswap Router to transfer "feeAmount" token from TRUSTED_DEPOSIT_TOKEN_ADDRESS. The Uniswap Router only transfers "buyBackFeeAmount" token.

Recommendation

We advise developers to check and make sure to use the correct amount of "buyBackFeeAmount."

Alleviation

The issue is resolved in commit "8bf1eea9f88be9d26bfc93311bdec0d503f1d405".

VAL-02 | function updateAccount called multi-time

Category	Severity	Location	Status
Gas Optimization	● Informational	vault.sol: 1157(Vault)	ⓘ Acknowledged

Description

In function "claim," four internal functions are being called, function "updateAccount" is called in each internal function, so "updateAccount" will be called four times, which will spend more gas.

Recommendation

We advise developers to call "updateAccount" only in the external "claimXXXDivs" function instead of the internal _claimXXXDivs function.

VAL-03 | Check-Return first to save gas

Category	Severity	Location	Status
Gas Optimization	● Informational	vault.sol: 1074(Vault), 1085(Vault), 1128(Vault)	ⓘ Acknowledged

Description

In functions _claimEthDivs, _claimTokenDivs, and _claimPlatformTokenDivs, they first update user reward balance to 0, and then check amount == 0. It's better to check and return first and then update the user reward balance.

Recommendation

we recommend checking if the amount equals zero first and then update user reward balance.

```
function _claimEthDivs() private {
        uint amount = ethDivsBalance[msg.sender];
        if (amount == 0) return;
        ethDivsBalance[msg.sender] = 0;
        decreaseTokenBalance(address(0), amount);
        msg.sender.transfer(amount);
        totalEarnedEthDivs[msg.sender] = totalEarnedEthDivs[msg.sender].add(amount);

        emit EtherRewardClaimed(msg.sender, amount);
}
```

VAL-04 | Return value not handled

Category	Severity	Location	Status
Volatile Code	● Informational	vault.sol: 1201(VaultWETH), 1241(VaultWETH)	ⓘ Acknowledged

Description

Variable holders, type of EnumerableSet.AddressSet, have their elements added and removed in state() , unstake() and emergencyUnstake(). The boolean return values of functions add(AddressSet, address) and remove(AddressSet, address) are not properly handled.

```
if (depositTokenBalance[msg.sender] == 0) {
            holders.remove(msg.sender);
        }
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

VAL-05 | 3rd Party Dependencies

Category	Severity	Location	Status
Control Flow	● Minor	vault.sol	ⓘ Acknowledged

Description

The contract is serving as the underlying entity to interact with third party Compound protocol and Uniswap protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume its functional correctness. However in the real world, 3rd parties may be compromised that led to assets lost or stolen.

Recommendation

We understand that the business logics of DYP earn vault require the interaction with Compound protocol and Uniswap protocol for the sake of pursuing capital gains of its users. We encourage the team to constantly monitor the statuses of those 3rd parties to mitigate the side effects when unexpected activities are observed.

VAL-06 | Transaction fee not handled in function emergencyWithdraw

Category	Severity	Location	Status
Logical Issue	● Minor	vault.sol: 1247~1287(Vault)	ⓘ Acknowledged

Description

When a user called function emergencyWithdraw() to withdraw their funds from Compound, he needs to pay a 0.3% withdraw fee. The withdraw fee does not distribute to other users as a reward. The withdraw fee remains in the contract, and only the contract owner can claim those fees.

Recommendation

We recommend handling transaction fees in function emergencyWithdraw by distributing the fee to all vault users.

VAU-01 | Missmatched Approve amount

Category	Severity	Location	Status
Inconsistency, Volatile Code	● Minor	vault-weth.sol: 1272~1279(VaultWETH)	✓ Resolved

Description

In function "handleFee," the contract called "safeApprove" allows Uniswap Router to transfer "feeAmount" token from TRUSTED_DEPOSIT_TOKEN_ADDRESS. The Uniswap Router only transfers "buyBackFeeAmount" token.

Recommendation

We advise developers to check and make sure to use the correct amount of "buyBackFeeAmount."

Alleviation

The issue is resolved in commit "8bf1eea9f88be9d26bfc93311bdec0d503f1d405".

VAU-02 | function updateAccount called multi-time

Category	Severity	Location	Status
Gas Optimization	● Informational	vault-weth.sol: 1165(Vault)	ⓘ Acknowledged

Description

In function "claim," four internal functions are being called, function "updateAccount" is called in each internal function, so "updateAccount" will be called four times, which will spend more gas.

Recommendation

We advise developers to call "updateAccount" only in the external "claimXXXDivs" function instead of the internal _claimXXXDivs function.

VAU-03 | Check-Return first to save gas

Category	Severity	Location	Status
Gas Optimization	● Informational	vault-weth.sol: 1082(Vault), 1093(Vault), 1138(Vault)	ⓘ Acknowledged

Description

In functions _claimEthDivs, _claimTokenDivs, and _claimPlatformTokenDivs, they first update user reward balance to 0, and then check amount == 0. It's better to check and return first and then update the user reward balance.

Recommendation

we recommend checking if the amount equals zero first and then update user reward balance.

```
function _claimEthDivs() private {
        uint amount = ethDivsBalance[msg.sender];
        if (amount == 0) return;
        ethDivsBalance[msg.sender] = 0;
        decreaseTokenBalance(address(0), amount);
        msg.sender.transfer(amount);
        totalEarnedEthDivs[msg.sender] = totalEarnedEthDivs[msg.sender].add(amount);

        emit EtherRewardClaimed(msg.sender, amount);
}
```

VAU-04 | Return value not handled

Category	Severity	Location	Status
Volatile Code	● Informational	vault-weth.sol: 1213(VaultWETH), 1255(VaultWETH)	ⓘ Acknowledged

Description

Variable holders, type of EnumerableSet.AddressSet, have their elements added and removed in state() , unstake() and emergencyUnstake(). The boolean return values of functions add(AddressSet, address) and remove(AddressSet, address) are not properly handled.

```
if (depositTokenBalance[msg.sender] == 0) {
            holders.remove(msg.sender);
        }
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

VAU-05 | 3rd Party Dependencies

Category	Severity	Location	Status
Control Flow	● Minor	vault-weth.sol	ⓘ Acknowledged

Description

The contract is serving as the underlying entity to interact with third party Compound protocol and Uniswap protocols. The scope of the audit would treat those 3rd party entities as black boxes and assume its functional correctness. However in the real world, 3rd parties may be compromised that led to assets lost or stolen.

Recommendation

We understand that the business logics of DYP earn vault require the interaction with Compound protocol and Uniswap protocol for the sake of pursuing capital gains of its users. We encourage the team to constantly monitor the statuses of those 3rd parties to mitigate the side effects when unexpected activities are observed.

VAU-06 | Transaction fee not handled in function emergencyWithdraw

Category	Severity	Location	Status
Logical Issue	● Minor	vault-weth.sol: 1261~1303(VaultWETH)	ⓘ Acknowledged

Description

When a user called function emergencyWithdraw() to withdraw their funds from Compound, he needs to pay a 0.3% withdraw fee. The withdraw fee does not distribute to other users as a reward. The withdraw fee remains in the contract, and only the contract owner can claim those fees.

Recommendation

We recommend handling transaction fees in function emergencyWithdraw by distributing the fee to all vault users.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

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

Magic Numbers

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

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK