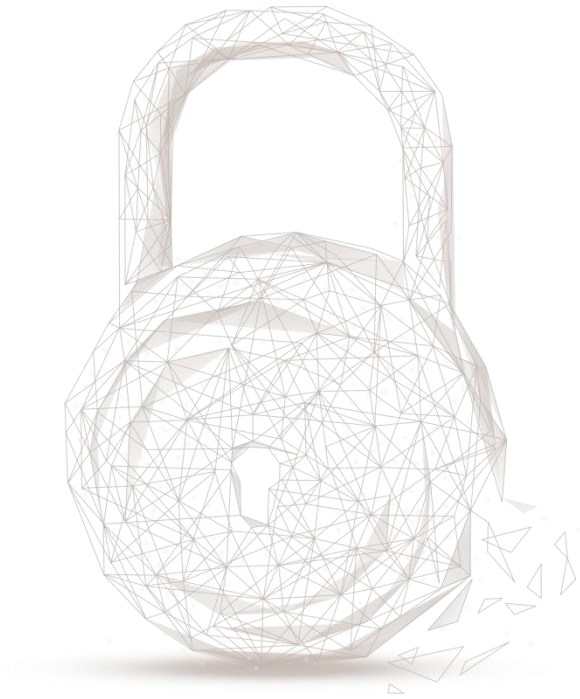
BEOSIN

Blockchain Security

Smart contract security audit report



Audit Number：202012281723

Report Query Name：SNP

Project Name：NervLedger-Vault

Smart Contract Information：

Contract name	Smart Contract Address	Smart Contract Address Link
controller	0xdaae16d48512b334fed0045816e58d89c401c934	https://etherscan.io/address/0xdaae16d48512b334fed0045816e58d89c401c934#code
sVault	0x6a71f460d90Eef13980A00C98481bBfA8316A51B	https://etherscan.io/address/0x6a71f460d90eef13980a00c98481bbfa8316a51b#code
StrategyDForceUSDT	0xeaae6108dcd7ffbcd8e2f86411628fad20aba41e	https://etherscan.io/address/0xeaae6108dcd7ffbcd8e2f86411628fad20aba41e#code
StrategyUSDT3pool	0x19a3e32de1518fdfde681cd64afca3a786a41c87	https://etherscan.io/address/0x19a3e32de1518fdfde681cd64afca3a786a41c87#code

Start Date：2020.12.21

Completion Date：2020.12.28

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass

		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass
		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Note: Audit results and suggestions in code comments

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract project SNP, including Coding Standards, Security, and Business Logic. **SNP contract passed all audit items. The overall result is Pass .** Please find below the basic information of the smart contract:

Business Audit:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.

- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 sVault Contract Audit

3.1.1 Basic token information of sVault

The contract implements a basic ERC20 token, and its basic information is as follows:

Token name	snp Tether USD
Token symbol	sUSDT
decimals	6
totalSupply	Initial supply is 0 (Mintable without cap; Burnable)
Token type	ERC20

Table 2 – Basic Token Information

3.1.2 sUSDT Token Functions

- Description: This contract token implements the basic functions of ERC20 standard tokens, and token holders can call corresponding functions for token transfer, approve and other operations.
- Related functions: *name, symbol, decimals, balanceOf, allowance, transfer, transferFrom, approve,*
- Safety Suggestion: Beware that changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. It is recommended that users reset the allowance to zero, and then set a new allowance.
- Result: **Pass**

3.1.3 sUSDT Token burning

- Description: The token holders can call the *withdraw* function(internally call *_burn* function) to destroy their sUSDT tokens.

- Related functions: *_burn, withdraw*
- Safety Suggestion: None
- Result: **Pass**

3.1.4 sUSDT Token minting

- Description: The contract implements the *deposit* function to call *_mint* function for issuing sUSDT tokens when users stake.
- Related functions: *_mint, deposit*
- Safety Suggestion: None
- Result: **Pass**

3.1.5 deposit function

- Description: The contract implements the *deposit* function for users to deposit USDT tokens. Users can obtain sUSDT tokens by calling this function to deposit their own USDT. When the USDT balance in the contract is greater than earnLowerlimit, this function will call the earn function to transfer 95%(can change, currently is 95% as default) of the USDT to the controller contract. And the earn function of the controller contract will be called at the same time.
- Related functions: *balance, deposit, earn, _mint*
- Safety Suggestion: None
- Result: **Pass**

3.1.6 withdraw function

- Description: The contract implements the *withdraw* function to extract the corresponding USDT reward. Users could obtain USDT rewards by calling _burn function to destroy their own sUSDT tokens. And the balance function is called to calculate the ratio to the total sUSDT tokens. When the USDT in the contract is not enough to pay the rewards obtained by the user, this function will transfer part of the USDT from the StrategyDForceUSDT contract to this contract, and charge a part of fee.
- Related functions: *balance, withdraw, _burn*
- Safety Suggestion: None
- Result: **Pass**

3.1.7 harvest function

- Description: The contract implements the *harvest* function to transfer the specified tokens (except USDT) of this contract to the controller contract, and required the caller is only the controller contract address.

- Related functions: *harvest*
- Safety Suggestion: There is no calling logic implementation of function *harvest*(require the caller to be controller contract address) in the whole controller contract. After confirming with the project side, this function is used to prevent users from wrongly sending other ERC20 tokens to this contract. Just changing the controller address to a user account address can call this function when it has use need.
- Result: **Pass**

3.1.8 Other setting functions

- Description: The contract implements the functions including *setMin, setGovernance, setController* and *setEarnLowerlimit* to update some parameters related to the project. The update of these settings may affect the user's income.
- Related functions: *setMin, setGovernance, setController, setEarnLowerlimit*
- Safety Suggestion: Adjusting fee will affect the user's income, it is recommended to set the maximum limit of the handling fee.None
- Fix Result: Ignored
- Result: **Pass**

3.2 Controller Contract Audit

3.2.1 earn function

- Description: The contract implements the *earn* function used to transfer tokens to the StrategyDForceUSDT contract to increase the amount of deposited USDT. Anyone can call this function, and this function calls the deposit function to increase the amount of deposited USDT by transferring tokens to the StrategyDForceUSDT address.
- Related functions: *earn, deposit*
- Safety Suggestion: None
- Result: **Pass**

3.2.2 yearn function

- Description: The contract implements the *yearn* function to obtain the specified tokens (not USDT and dUSDT) in the StrategyDForceUSDT contract and convert them to USDT. Only the addresses strategist and government can call the *yearn* function. This function obtains all the specified token balances in the StrategyDForceUSDT by calling the withdraw function, and calls the *swap* function to convert the tokens into USDT, and finally calls the *earn* function to transfer the converted USDT to the StrategyDForceUSDT contract for increasing deposited amount, this operation will charge 5% handling fee and it can be adjusted by the project party.

- Related functions: *withdraw, swap, earn, getExpectedReturn*
- Safety Suggestion: None.
- Result: **Pass**

3.2.3 withdrawAll, inCaseTokensGetStuck, inCaseTokensGetStuck functions

- Description: The contract implements *withdrawalAll*, *inCaseTokensGetStuck* and *inCaseTokensGetStuck* to withdraw specified tokens. Only the strategist and government addresses can call these three functions. The *withdrawAll* function withdrtaws all the USDT owned by this contract to the sVault contract by calling the *withdrawAll* function of the StrategyDForceUSDT contract. The *inCaseTokensGetStuck* function can withdraw all tokens of this contract to the caller's address. The *inCaseStrategyTokenGetStuck* function can withdraw the specified tokens (not USDT and dUSDT) in StrategyDForceUSDT to this contract.

- Related functions: *withdrawAll, inCaseTokensGetStuck, inCaseTokensGetStuck*
- Safety Suggestion: None
- Result: **Pass**

3.2.4 Other setting functions

- Description: The contract implements the functions including *setRewards*, *setStrategist, setSplit, setOneSplit, setGovernance, setVault, setConverter* and *setStrategy* to update some parameters related to the project. The update of these settings may affect the user's income.
- Related functions: *setRewards, setStrategist, setSplit, setOneSplit, setGovernance, setVault, setConverter, setStrategy*
- Safety Suggestion: Adjusting fee will affect the user's income, it is recommended to set the maximum limit of the handling fee.None
- Fix Result: Ignored
- Result: **Pass**

3.3 StrategyDForceUSDT Contract Audit

3.3.1 deposit function

- Description: The contract implements the *deposit* function to increase the deposited token amount in to the pool contract. This function can be called by anyone and the function can obtain dUSDT tokens by calling the mint function, and then obtains DForce token rewards by depositing dUSDT tokens into the pool contract.

- Related functions: *stake, deposit, mint*
- Safety Suggestion: None
- Result: Pass

3.3.2 withdraw functions

- Description: The contract implements *withdraw*(IERC20 _asset), *withdraw*(uint256 _amount), *withdrawal* functions to withdraw tokens in the contract. Only the controller address can call these three functions. The *withdraw*(IERC20 _asset) function can withdraw specified tokens (not USDT and dUSDT) to the controller address. The function *Withdraw*(uint256 _amount) is used to withdraw USDT tokens in this contract, and part of the handling fee will be charged to the rewards address when withdrawing. The *withdrawAll* function obtains the deposited tokens and reward tokens in the pool contract by calling the internal function *_withdrawAll*, and converts them into USDT, and finally withdraws them to the sVault contract.

- Related functions: *withdraw, withdrawsome, _withdrawAll*
- Safety Suggestion: None
- Result: **Pass**

3.3.3 harvest function

- Description: The contract implements the harvest function to get the reward tokens in the pool contract. The function obtains reward tokens by calling the *getReward* function, and converts the reward tokens into USDT through the interface function of uniswap, and finally transfers 50%(can be controlled by governance address) of the rewarded USDT to the rewards address of the project party, and the remaining part will be staked into the reward pool contract by calling the deposit function.

- Related functions: *harvest, getReward, swapExactTokensForTokens*
- Safety Suggestion: None
- Result: **Pass**

3.3.4 _withdrawSome function

- Description: The contract implements the *_withdrawSome* function to withdraw part of the staked tokens in the reward pool contract. The function withdraws part of the staked dUSDT tokens by calling the withdraw function, and then calls the redeem function to convert them into USDT and transfer to this contract.

- Related functions: *_withdrawSome, redeem, withdraw*
- Safety Suggestion: None
- Result: **Pass**

3.3.5 Other setting functions

- Description: The contract implements the functions including *setStrategist, setWithdrawalFee, setPerformanceFee and setGovernance* to update some parameters related to the project. The update of these settings may affect the user's income.
- Related functions: *setStrategist, setWithdrawalFee, setPerformanceFee, setGovernance*
- Safety Suggestion: Adjusting fee will affect the user's income, it is recommended to set the maximum limit of the handling fee.
- Fix Result: Ignored
- Result: **Pass**

3.4 StrategyUSDT3pool Contract Audit

3.4.1 deposit function

- Description: The contract implements the *deposit* function to increase the deposited amount in the y3crv contract. This function can only be called after the specified address met the check condition in *isAuthorized* modifier, and the handling fee will be charged during the function call. The function uses USDT to add liquidity in the 3pool contract to obtain 3crv tokens and then calls the *deposit* function of the y3crv contract to stake the 3crv tokens to get rewards.
- Related functions: *rebalance, deposit, drip, trik*
- Safety Suggestion: None
- Result: Pass

3.4.2 withdraw functions

- Description: The contract implements *withdraw*(IERC20 _asset), *withdraw*(uint256 _amount), *withdrawAll*() functions to withdraw tokens in the contract. Only the controller address can call these three functions. The *withdraw*(IERC20 _asset) function can withdraw the specified token (not USDT, 3crv, y3crv) to the controller address. The *withdraw*(uint256 _amount) is used to withdraw USDT tokens in the contract, and part of the handling fee will be charged to the rewards address when withdrawing (the handling fee can be adjusted). The *withdrawAll* function obtains 3crv tokens by calling the internal function *_withdrawAll*, then calls the *_withdrawOne* function to convert 3crv tokens to USDT, and finally withdraw them to the sVault contract.
- Related functions: *withdraw,_ withdrawsome, _ withdrawone, withdrawAll*
- Safety Suggestion: None
- Result: Pass

3.4.3 forceD function

- Description: The contract implements the *forceD* function to increase the deposited amount in the y3crv contract. This function can only be called after the specified address met the check condition in

isAuthorized modifier. The function uses USDT to add liquidity in the 3pool contract to obtain 3crv tokens and then calls the *deposit* function of the y3crv contract to stake the 3crv tokens to get rewards.

- Related functions: *add_liquidity, forceD, deposit*
- Safety Suggestion: None
- Result: Pass

3.4.4 forceW function

- Description: The contract implements the *forceW* function to withdraw staked 3crv tokens. This function can only be called after the specified address met the check condition in *isAuthorized* modifier. The function withdraws a specified number of 3crv tokens to this contract by calling the *withdraw* function, and then converts the 3crv tokens into USDT to this contract by calling the remove_liquidity_one_coin function.

- Related functions: *remove_liquidity_one_coin, forceW, withdraw*
- Safety Suggestion: None
- Result: Pass

3.4.5 drip function

- Description: The contract implements the *drip* function for charging fees. This function can only be called after the specified address met the check condition in *isAuthorized* modifier. The function will transfer 0.5% (adjustable) of the handling fee to the strategist address and 5% (adjustable) of the handling fee to the rewards address.

- Related functions: *drip, trik, getPricePerFullShare, get_virtual_price*
- Safety Suggestion: None
- Result: Pass

3.3.6 Other setting functions

- Description: The contract implements the functions including *setGovernance, setStrategist, setWithdrawalFee, setTreasuryFee, setStrategistReward, setThreshold* and *setSlip* to update some parameters related to the project. The update of these settings may affect the user's income.

- Related functions: *setGovernance, setStrategist, setWithdrawalFee, setStrategistReward, setThreshold, setSlip, setTreasuryFee*
- Safety Suggestion: Adjusting fee will affect the user's income, it is recommended to set the maximum limit of the handling fee.
- Fix Result: Ignored
- Result: **Pass**

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts project SNP. The SNP project passed all audit items. In this project, all found issues have been informed to the project side. There being an issue that the handling fee can be adjusted easily by the governance address, the users' income could be inconsistent with the expected value. After communicating with the project party, this issue is ignored. The overall audit result is Pass.

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com