Gennix Smart Contract Security Audit Report Abstract

Customer: TheGennix
Website: https://www.gennix.io/
Platform: Binance Smart Chain
Language: Solidity
Date: September 9th, 2021

Claimed Smart Contracts Features

Claimed Feature Detail	Our Observation
File 1: AdminRole.sol ● The Adminrole owner can access functionality like : add new admin and remove admin.	**YES, This is valid.**
File 2: FarmingInterface.sol ● The FarmingInterface can access functions like: totalAllocPoint, poolsCount, rewardPerBlock, etc.	**YES, This is valid.**
File 3: FarmingLens.sol ● The FarmingLens owner can access functions like:getBalanceAll, getMetadataAll, etc.	**YES, This is valid.**
File 4: FarmingStorage.sol ● The FarmingStorage can store pool details, LP etc.	**YES, This is valid.**

File 5: FeeToken.sol

YES, This is valid.

- The FeeToken can set fees and treasury addresses.

File 6: GennixToken_old.sol

YES, This is valid.

- Name: Gennix Token
- Symbol: GNNX
- Decimals: 8

File 7: LPToken.sol

YES, This is valid.

- Name: Pancake LPs
- Symbol: Cake-LP
- Decimals: 12

File 8: MintFarming.sol

YES, This is valid.

- MintFarming can add a new LP, Update the given pool's Reward allocation point, etc.

File 9: Whitelist.sol

YES, This is valid.

- The Whitelist owner can add and remove wallet addresses from whitelist.

File 10: Stacking.sol

YES, This is valid.

- Stage Interval: 10 minutes

File 11: StackingErc20.sol

YES, This is valid.

- The StackingErc20 can access functions like: transfer, transferFrom.

File 12: StackingInterface.sol

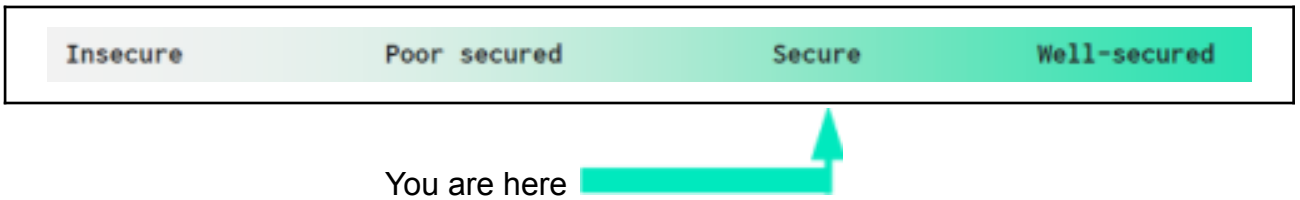
YES, This is valid.

- The StackingInterInterface file is empty.

File 13: AdminRole.sol	YES, This is valid.
● The Adminrole owner can access functionality like : add new admin and remove admin, check admin.	
File 14: GennixToken.sol	**YES, This is valid.**
● Name: Gennix Token ● Symbol: GNNX ● Decimals: 8	

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



We used various tools like MythX, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 low and some very low-level issues.

Technical Quick Stats

Main Category	Subcategory	Result
Contract	Solidity version not specified	Passed

Programming	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Moderated
	Critical operation lacks event log	Passed
	Random number generation/use vulnerability Fallback function misuse	Passed Passed
	Race condition Logical vulnerability	Passed Passed
	Features claimed Other programming issues	Passed Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

These audit scope have 24 smart contracts. These smart contracts also contain Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in the TheGennix contracts are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the TheGennix contracts.

The team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Some code parts are not well commented on smart contracts.

Documentation

We were given The Gennix smart contracts code in the form of a hash code. The details of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So, it is difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well-known industry standard open-source projects. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

AdminRole.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	internal	Passed	No Issue
2	onlyAdmin	mdifier	Passed	No Issue
3	isAdmin	read	Passed	No Issue
4	addAdmin	write	access only Admin	No Issue
5	renounceAdmin	write	Passed	No Issue
6	_addAdmin	internal	Passed	No Issue
7	_removeAdmin	internal	Passed	No Issue

FarmingInterface.sol

Sl.	Functions	Type	Observation	Conclusion
1	getPendingReward	read	Passed	No Issue
2	totalAllocPoint	external	Passed	No Issue
3	rewardPerBlock	external	Passed	No Issue
4	tokens	external	Passed	No Issue
5	poolsCount	external	Passed	No Issue

FarmingLens.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	getBalanceOf	external	Passed	No Issue
3	getBalanceAll	external	Passed	No Issue
4	getMetadataOf	external	Passed	No Issue
5	getMetadataAll	external	Passed	No Issue
6	getLPMetadata	external	Passed	No Issue
7	getLPMetadataAll	external	Passed	No Issue
8	getTokens	external	Passed	No Issue

FarmingStorage.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	transfer	write	Passed	No Issue

3	transferFrom	internal	Passed	No Issue
4	setFee	write	access only Admin	No Issue
5	setPendingTreasury	write	access only Admin	No Issue
6	acceptTreasury	write	Passed	No Issue

LPToken.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue

MintFarming.sol

Sl.	Functions	Type Observation	Observation	Conclusion
1	init	external	Passed	No Issue
2	poolsCount	external Passed	Passed	No Issue
3	_requirePoolNotExists	internal Passed	Passed	No Issue
4	_registerPool	internal Passed	Passed	No Issue
5	addToken	write	Critical operation lacks event log	Refer audit findings section below
6	setAllocPoint	write Critical operation lacks event log	Critical operation lacks event log	Refer audit findings section below
7	setRewardAddress	write Critical operation lacks event log	Critical operation lacks event log	Refer audit findings section below
8	_updateStakingPool	internal	Infinite loop	Refer audit findings section below
9	getMultiplier	read	Passed	No Issue
10	getPendingReward	read	Passed	No Issue
11	massUpdatePools	write	Infinite loop	Refer audit findings section below

12	updatePool	write	Critical operation lacks event log	Refer audit findings section below
13	deposit	write	Passed	No Issue
14	withdraw	write	Passed	No Issue
15	emergencyWithdraw	write	Passed	No Issue
16	_reward	write	Passed	No Issue

Whitelist.sol

Sl.	Functions	Type	Observation	Conclusion
1	addWhitelist	write	access only Admin	No Issue
2	removeWhitelist	write	access only Admin	No Issue
3	isWhitelisted	internal	Passed	No Issue

Stacking.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	deposit	external	Passed	No Issue
3	claim	external	Passed	No Issue
4	withdraw	external	Critical operation lacks event log	Refer audit findings section below
5	addReward	external	Critical operation lacks event log	Refer audit findings section below
6	rewardOf	read	Passed	No Issue
7	_innerClaimWithChecks	internal	Passed	No Issue
8	_innerAddReward	internal	Passed	No Issue
9	_innerDepositNoSend	internal	Passed	No Issue
10	_innerDeposit	internal	Passed	No Issue
11	_innerWithdrawNoSend	internal	Passed	No Issue
12	_innerWithdraw	internal	Passed	No Issue
13	_calculateReward	internal	Passed	No Issue
14	_innerClaim	internal	Passed	No Issue
15	getStateTs	write	Passed	No Issue
16	getCurrentStateTs	read	Passed	No Issue
17	_transferUnderlying	internal	Passed	No Issue
18	_transferUnderlyingFrom	internal	Passed	No Issue
19	_allowanceUnderlying	internal	Passed	No Issue

StackingErc20.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	transfer	write	Passed	No Issue
3	transferFrom	write	Passed	No Issue

StackingStorage.sol

Sl.	Functions	Type	Observation	Conclusion
1	totalSupply	read	Passed	No Issue
2	balanceOf	read	Passed	No Issue
3	transfer	write	Passed	No Issue
4	allowance	read	Passed	No Issue
5	approve	write	Passed	No Issue
6	transferFrom	write	Passed	No Issue
7	increaseAllowance	write	Passed	No Issue
8	decreaseAllowance	write	Passed	No Issue
9	_transfer	internal	Passed	No Issue
10	_mint	internal	Passed	No Issue
11	_burn	internal	Passed	No Issue
12	_approve	internal	Passed	No Issue
13	_burnFrom	internal	Passed	No Issue

ERC20Burnable.sol

Sl.	Functions	Type	Observation	Conclusion
1	burn	write	Passed	No Issue
2	burnFrom	write	Passed	No Issue

ERC20Detailed.sol

Sl.	Functions	Type	Observation	Conclusion
1	name	read	Passed	No Issue
2	symbol	read	Passed	No Issue
3	decimals	read	Passed	No Issue

ERC20Mintable.sol

Sl.	Functions	Type	Observation	Conclusion
1	mint	write	access only Minter	No Issue

AdminRole.sol

Sl.	Functions	Type	Observation	Conclusion
1	onlyAdmin	modifier	Passed	No Issue
2	isAdmin	read	Passed	No Issue
3	addAdmin	write access only Admin	Passed	No Issue
4	renounceAdmin	write	Passed	No Issue
5	_addAdmin	internal	Passed	No Issue
6	_removeAdmin	internal	Passed	No Issue

GennixToken.sol

Sl.	Functions	Type	Observation	Conclusion
1	init	write	Passed	No Issue

MinterRole.sol

Sl.	Functions	Type	Observation	Conclusion
1	onlyMinter	modifier	Passed	No Issue
2	isMinter	read	Passed	No Issue
3	addMinter	write access only Admin	Passed	No Issue
4	renounceMinter	write	Passed	No Issue
5	_addMinter	internal	Passed	No Issue
6	_removeMinter	internal	Passed	No Issue

Ownable.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	isOwner	read	Passed	No Issue
5	renounceOwnership	write	access only Owner	No Issue
6	transferOwnership	write	access only Owner	No Issue
7	_transferOwnership	internal	Passed	No Issue

Token.sol

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	write	Passed	No Issue
3	onlyOwner	internal	Passed	No Issue
4	isOwner	write	access only Admin	No Issue
5	setPendingTreasury	write	access only Admin	No Issue
6	acceptTreasury	write	Passed	No Issue

TokenStorage.sol

Sl.	Functions	Type	Observation	Conclusion
1	addWhitelist	write	access only Admin	No Issue
2	removeWhitelist	write	access only Admin	No Issue
3	isWhitelisted	internal	Passed	No Issue

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Infinite loop - (MintFarming.sol, FarmingLens.sol)

```
function _updateStakingPool() internal {
    uint256 length = tokens.length;
    uint256 points = 0;
    for (uint256 pid = 1; pid < length; ++pid) {
        points = points.add(poolInfo[address(tokens[pid])].allocPoint);
    }
    if (points != 0) {
        totalAllocPoint = totalAllocPoint.sub(poolInfo[address(tokens[0])].allocPoint).add(points);
        poolInfo[address(tokens[0])].allocPoint = uint128(points);
    }
}
```

```
// Update reward variables for all pools. Be careful of g
function massUpdatePools() public {
    uint256 length = tokens.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(tokens[pid]);
    }
}
```

There are several places in the smart contracts, where array.length is used directly in the loop. It is recommended to put some kind of limits.
Resolution: Adjust logic to replace loops with mapping or other code structures. **Status:** Open

Very Low / Discussion / Best practices:

(1) Use the latest solidity version - (All contracts)

```
pragma solidity ^0.5.16;
```

Using the latest solidity will prevent any compiler-level bugs.
Resolution: Please use 0.8.7 which is the latest version.

Status: Open

(2) All functions which are not called internally, must be declared as external. It is more efficient as sometimes it saves some gas.
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices
Status: Open

(3) Critical operation lacks event log

There are several places in the smart contracts, were not added a critical function call event log.
Resolution: Below functions should log events.

MintFarming.sol - addToken, setAllocPoint, setRewardAddress, updatePool Stacking.sol - withdraw, addReward
Status: Open

Centralization

These smart contracts have some functions which can be executed by Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- addToken: The MintFarming owner can add a new token.
- setAllocPoint: The MintFarming owner can set allocated points.
- setRewardAddress: The MintFarming owner can set reward wallet addresses. ● addWhitelist: The Whitelist Owner can add wallet addresses in white list. ● removeWhitelist: The Whitelist Owner can remove wallet addresses from whitelist. ● mint: The ERC20Mintable owner can create `amount` tokens and assign them to `account`, increasing the total supply.
- addAdmin: The AdminRole admin can add a new admin owner address.
- addMinter: The MinterRole admin can add minter.
- renounceOwnership: The Ownable owner can renounce ownership.
- transferOwnership: The Ownable owner can transfer ownership.
- setFee: The Token owner can set a fee.
- setPendingTreasury: The Token owner can set the pending Treasury.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues, but they are not critical. So, **it's good to go to production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.
Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured".**