PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Genesys - DNA Protocol

03 December 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for DNA Protocol on the Fantom Opera network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Genesys - DNA Protocol
URL	https://genesys.app/
Platform	Fantom Opera
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
DNA	0xF8b234A1CE59991006930de8B0525f9013982746	MATCH
MasterChef	0x18cD511b4ad613308Bd0C795e85Fbd8BE1a0aF94	MATCH
Timelock	0x1B8a2b8Db5d08fDC7E9245aA0a8c91900f4736A4	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	1	1	-	-
Low	3	3	-	-
Informational	7	6	-	1
Total	**13**	**12**	**1**	**1**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 DNA

ID	Severity	Summary	Status
01	HIGH	`updategenesysSwapRouter` could be used to siphon all liquidity fees or turn the token into a honeypot	RESOLVED
02	HIGH	LP tokens from `addLiquidity` are sent to operator	RESOLVED
03	LOW	mint function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
04	LOW	FTM dust after adding liquidity will accumulate in the contract and not be retrievable	RESOLVED
05	INFO	Operator address is set to `private`	RESOLVED
06	INFO	Unnecessary updating of `amount` in `_transfer` to `sendAmount`	RESOLVED
07	INFO	`_totalSupply` is sufficient to keep track of the total token supply minted	RESOLVED

1.3.2 MasterChef

ID	Severity	Summary	Status
08	MEDIUM	Lack of maximum upper limit for setting `dnaPerSecond`	RESOLVED
09	LOW	`pendingDna` will show inaccurate pending harvests on the dapp frontend if the pending rewards causes `totalSupply` to be exceed `MAXSUPPLYCAP`	RESOLVED
10	INFO	Total token supply might not be minted due to try and catch pattern	RESOLVED
11	INFO	SafeMath is unnecessary as Solidity compiler 0.8.7 is used	ACKNOWLEDGED
12	INFO	`setNextBoost` unnecessarily calls `massUpdatePools` more than once	RESOLVED
13	INFO	`userPoolLockup` logic can be simplified	RESOLVED

1.3.3 Timelock

No issues found.

2 Findings

2.1 DNA

The DNA token is a simple ERC-20 token which will be used as the main reward token for the Masterchef. It allows for DNA tokens to be minted when the `mint` function is called by the owner of the contract, which at the time of deployment would be the DNAProtocol team. Ownership of the contract is generally transferred to the Masterchef after deployment. The token has a maximum supply of 11,000. 500 tokens are preminted for initial liquidity.

There is transfer tax of up to maximum of 5%, which is imposed on all transfers except under the following conditions:

- Recipient is burn address
- Sender is token contract owner
- Recipient is token contract owner
- Sender is operator
- Recipient is operator

A portion of the transfer tax will be sent to the burn address if the `burnRate` is activated. The remaining transfer tax amount will be used for liquidity.

2.1.1 Token Overview

Address	0xF8b234A1CE59991006930de8B0525f9013982746
Token Supply	11,000
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None
Pre-mints	500

2.1.2 Privileged Roles

The following functions can be called by the owner of the contract:

- `mint`
- `transferOwnership`
- `renounceOwnership`
- `updategenesysSwapRouter`

The following functions can be called by the operator of the contract:

- `updateTransferTaxRate`
- `updateBurnRate`
- `updateMinAmountToLiquify`
- `updateSwapAndLiquifyEnabled`
- `transferOperator`

2.1.3 Issues & Recommendations

Issue #01	**`updategenesysSwapRouter` could be used to siphon all liquidity fees or turn the token into a honeypot**
Severity	HIGH SEVERITY
Description	The owner can update the router that generates liquidity to an address or contract of choice. This contract could be a malicious contract that simply keeps the tokens sent to it or even worse a contract that presents selling transactions by always reverting.
Recommendation	Consider allowing this function to be called only once. `require(genesysSwapRouter == address(0), "Router set already");` If this is not possible, consider a significantly longer timelock as the owner so that users can have sufficient time to react to future router changes.
Resolution	RESOLVED The check to ensure that the current `genesysSwapRouter` is the zero address before allowing it to be set is added.

Issue #02	LP tokens from `addLiquidity` are sent to operator
Severity	HIGH SEVERITY

Location	Line 982~
	`genesysSwapRouter.addLiquidityETH{value: ethAmount}(`
	`address(this),`
	`tokenAmount,`
	`0, // slippage is unavoidable`
	`0, // slippage is unavoidable`
	`operator(),`
	`block.timestamp`
	`);`

Description	The LP tokens provided from the `addLiquidity` are sent to the operator, who will be able to decompose the LPs and obtain the underlying tokens.
Recommendation	Consider setting the `to` address for adding of liquidity to a burn address, ensuring that the received liquidity pool tokens are locked forever.
Resolution	RESOLVED LP tokens are now sent to the `BURN_ADDRESS`.

Issue #03	**`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint tokens for dumping.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED Ownership of the token has been transferred to the Masterchef.

Issue #04	**FTM dust after adding liquidity will accumulate in the contract and not be retrievable**
Severity	LOW SEVERITY
Description	When DNA/FTM liquidity is added, there could be some FTM which is leftover as dust. Subsequent adding of liquidity does not use this FTM balance. This FTM balance would grow over time and be stuck in the token contract as there is no way to transfer the FTM out from the token contract.
Recommendation	Consider adding an `onlyOperator` access controlled function to allow withdrawing of any FTM balances in the token contract.
Resolution	RESOLVED A `withdrawStuckToken` that allows the operator to withdraw any ERC20 token except DNA has been added.

Issue #05	Operator address is set to `private`
Severity	INFORMATIONAL
Location	Line 855 `address private _operator;`
Description	The `_operator` state variable is set to private, which does not allow for easy visibility of the operator's address. Ideally, the operator should be a timelock which has a reasonable delay for making sensitive state changes to the token contract. Thus, if the operator address is not easily readable, it would make it harder for users to verify this.
Recommendation	Set the operator address to `public`.
Resolution	RESOLVED The operator address is now `public`.

Issue #06	Unnecessary updating of `amount` in `_transfer` to `sendAmount`
Severity	INFORMATIONAL

Location	Line 919~
	`super._transfer(sender, BURN_ADDRESS, burnAmount);`
	`super._transfer(sender, address(this), liquidityAmount);`
	`super._transfer(sender, recipient, sendAmount);`
	`amount = sendAmount;`

Description	After all the `super._transfer` calls are done, `amount` is set to `sendAmount`, but is not used after that.
Recommendation	Remove the `amount = sendAmount` statement as it is unnecessary to reduce gas costs.
Resolution	RESOLVED The unnecessary statement has been removed.

Issue #07	**`_totalSupply` is sufficient to keep track of the total token supply minted**
Severity	INFORMATIONAL
Description	Both `MAXCAP` and `_totalSupply` are incremented by the amount of tokens minted. There is no `burn` function, so the token supply will not decrease. This makes `MAXCAP` redundant as it will always be the same value as `_totalSupply`.
Recommendation	Consider removing `MAXCAP`, and keeping track of the token supply using `_totalSupply`.
Resolution	RESOLVED `MAXCAP` has been removed.

2.2 MasterChef

The DNAProtocol Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking the latter is the removal of the `migrator` function from the original Pancakeswap, which can be used maliciously by the team to steal user's tokens. DNAProtocol has limited the deposit fee to at most 4%, and a harvest interval that is capped to a maximum of 2 hours.

2.2.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `addInitialPools`
- `add`
- `set`
- `updateEmissionRate`
- `updateStartBlock`
- `transferOwnership`
- `renounceOwnership`

The following functions can be called by the operator of the contract:

- `setAllocPoints`
- `setNextBoost`

The following functions can be called by the DevAddr:

- `setDevAddr`

The following functions can be called by the FeeAddr:

- `setFeeAddr`

2.2.2 Issues & Recommendations

Issue #08	Lack of maximum upper limit for setting dnaPerSecond
Severity	MEDIUM SEVERITY
Description	There is a lack of any maximum cap as the `MAX_EMISSION_RATE` and its check in `updateEmissionRate` have been commented out. The owner can set the `dnaPerSecond` to any value, and possibly cause a sudden massive increase in emissions.
Recommendation	Consider uncommenting out the `MAX_EMISSION_RATE` and its check to enforce a maximum limit on the emission rate.
Resolution	RESOLVED A `MAX_EMISSION_RATE` of 0.02 tokens has been enforced in the setter.

Issue #09	**pendingDna will show inaccurate pending harvests on the dapp frontend if the pending rewards causes `totalSupply` to be exceed `MAXSUPPLYCAP`**
Severity	LOW SEVERITY
Description	Similarly to `updatePool`, `pendingDna` does not check if the pending rewards will cause the total supply to exceed the `MAXSUPPLYCAP`. This can cause inaccurate pending harvests to be shown towards the end of token emissions.

Recommendation	Consider factoring in the `MAXSUPPLYCAP`, and set the pending reward to be the difference between `MAXSUPPLYCAP` and `totalSupply` if the pending reward causes `totalSupply` to exceed `MAXSUPPLYCAP`.
	
	`uint256 dnaReward = multiplier * dnaPerSecond * pool.allocPoint / totalAllocPoint;`
	
	`if (dna.totalSupply().add(dnaReward) > dna.maxSupply()) {`
	`  dnaReward = dna.maxSupply() - dna.totalSupply();`
	`}`
	`accDnaPerShare = accDnaPerShare+(dnaReward * 1e18 / pool.lpSupply);`

Resolution	RESOLVED The recommendation has been implemented.

Issue #10	Total token supply might not be minted due to try and catch pattern
Severity	INFORMATIONAL

Description	As there is a `MAXCAPSUPPLY` for the DNA token, minting the reward and causing the maximum cap to exceed would result in a revert.
	
	`DNA Token::Line 757`
	`require(MAXCAP + amount <= MAXCAPSUPPLY, "Max supply reached");`
	
	To prevent this, the following try and catch pattern is done in `updatePool`.
	
	`Line 1571~`
	`try dna.mint(devaddr, dnaReward.div(10)) {`
	`} catch (bytes memory reason) {`
	`  dnaReward = 0;`
	`  emit DnaMintError(reason);`
	`}`
	
	`try dna.mint(address(this), dnaReward) {`
	`} catch (bytes memory reason) {`
	`  dnaReward = 0;`
	`  emit DnaMintError(reason);`
	`}`
	
	In the case where `totalSupply + amount` does exceed `MAXCAPSUPPLY`, the mint will not be done. This means that the token supply could be capped at an amount slightly lower than `MAXCAPSUPPLY`.

Recommendation	Consider minting the difference between `MAXCAPSUPPLY` and `totalSupply`, if any.

```
uint256 dnaReward = multiplier * dnaPerSecond * pool.allocPoint / totalAllocPoint;
uint256 devReward = dnaReward/10;
uint256 totalRewards = dna.totalSupply() + devReward + dnaReward;

if (totalRewards  <= dna.maxSupply()) {
    //  mint dev reward as normal as not at maxSupply
    dna.mint(devaddr, devReward);
} else {
    // update dnaReward to difference
    dnaReward= dna.maxSupply() - dna.totalSupply();
}

if (dnaReward != 0) {
    // only mint to MC and calculate and update accDnaPerShare if dnaReward is non 0
    dna.mint(address(this), dnaReward);
    pool.accDnaPerShare = pool.accDnaPerShare + (dnaReward * 1e18 / pool.lpSupply);
}

pool.lastRewardBlock = block.timestamp;
```

Resolution	RESOLVED The recommendation has been implemented.

Issue #11	**SafeMath is unnecessary as Solidity compiler 0.8.7 is used**
Severity	INFORMATIONAL
Description	Smart contracts compiled with Solidity 0.8.0 and above have built in integer overflow and underflow prevention.
Recommendation	Consider refactoring all usages of SafeMath to reduce the contract byte code and optimize for gas usage.
Resolution	ACKNOWLEDGED

Issue #12	`setNextBoost` unnecessarily calls `massUpdatePools` more than once
Severity	INFORMATIONAL

Location	Line 1466~:
	`function setNextBoost() external onlyOperator {`
	`  setAllocPoints(0, poolInfo[0].allocPoint + 1000);`
	`  setAllocPoints(1, poolInfo[1].allocPoint + 1000);`
	`  […]`

Description	`setNextBoost` is a function which allows the operator to call `setAllocPoints` for a hardcoded set of pool ids. Each time `setAllocPoints` is called, `massUpdatePools` is run, and all pools will be loop through to call `updatePool` on each pool. As `setNextBoost` has 14 pools hardcoded, calling it would call `massUpdatePools` 14 times, looping through at least 14 pools each time. This would result in unnecessary gas wastage.

Recommendation	Consider adding a bool `_withUpdate` parameter for `setAllocPoints`, similarly to how set has. Only the first call to `setAllocPoints` will require the calling of `massUpdatePools`.
	
	`function setNextBoost() external onlyOperator {`
	`  setAllocPoints(0, poolInfo[0].allocPoint + 1000, true);`
	`  setAllocPoints(1, poolInfo[1].allocPoint + 1000, false);`
	`  [...]`
	
	`function setAllocPoints(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOperator {`
	`  if (_withUpdate) {`
	`  massUpdatePools();`
	`  }`
	`  totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);`
	`  poolInfo[_pid].allocPoint = _allocPoint;`
	`}`

Resolution	RESOLVED The recommendation has been implemented.

Issue #13	`userPoolLockup` logic can be simplified
Severity	INFORMATIONAL
Description	`userPoolLockup` parses `user.nextHarvestUntil` and `block.timestamp`, both uint256 data types, into int data types for subtraction, and returns the result of subtraction of `user.nextHarvestUntil - block.timestamp`, or 0 if it is negative. This is unnecessary and can be simplified.
Recommendation	Consider refactoring to use this function instead: `function userPoolLockup(uint256 _pid, address _user) external view returns (uint256 lock) { UserInfo storage user = userInfo[_pid][_user]; lock = user.nextHarvestUntil <= block.timestamp ? 0 : user.nextHarvestUntil - block.timestamp; }`
Resolution	RESOLVED The recommendation has been implemented.

2.3 Timelock

The Timelock contract is a clean fork of Compound Finance's timelock. This is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools. This contract should be the owner of the Masterchef contract to time delay making sensitive changes such as adding a new pool, or changing the allocation for an existing pool.

Parameter	Value	Description
Delay	TBC	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	4 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.3.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY