GA GUARDIAN

G8Keep

Token Launchpad

Security Assessment

July 29th, 2024

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Wafflemakr, Mark Jonathas, Osman Ozdemir, Michael Lett, Darren Jensen

Client Firm G8Keep

Final Report Date July 29, 2024

Audit Summary

G8Keep engaged Guardian to review the security of its token launchpad. From the 8th of July to the 15th of July, a team of 7 auditors reviewed the source code in scope. All findings have been recorded in the following report.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Ethereum, Base**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/gatekeep-fuzzing

Table of Contents

Project Overview

Project Summary

Project Name	G8Keep
Language	Solidity
Codebase	https://github.com/g8keep/audit
Commit(s)	9a0fb9beec4e822807f860d9b14b7c6544d2e0c6

Audit Summary

Delivery Date	July 29, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	1	0	0	0	0	1
High	3	0	0	2	0	1
Medium	10	0	0	2	0	8
Low	22	0	0	4	0	18

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 Critical	Easily exploitable by anyone, causing loss/manipulation of assets or data.
🟠 High	Arduously exploitable by a subset of addresses, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

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.
- Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of G8Keep, fuzz-testing with Echidna was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared Echidna fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
GK-01	When buying g8keepTokens during snipe protection period adjustedAmountOut must be less than uniswapAmountOut (amount1Out)	✅	✅	N/A	10M+
GK-01R	When buying g8keepTokens during snipe protection period adjustedAmountOut must be less than or equal to uniswapAmountOut (amount1Out)	✅	N/A	✅	10M+
GK-02	adjustedAmountOut < balance1 - minimumToken1Balance (If buy amount exceeds maxSnipeProtectionBuyWithoutPenalty users should be penalized)	✅	❌	N/A	10M+
GK-02R	adjustedAmountOut < balance1 - cachedThirdPartyLPAmount (If buy amount exceeds maxSnipeProtectionBuyWithoutPenalty users should be penalized)	✅	N/A	✅	10M+
GK-03	When buying tokens and amountOutExpected is less than maxSnipeProtectionBuyWithoutPenalty amountOutAdjusted == uniswapAmountOut (amount1Out)	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
GK-03R	Cached third Party LP amount should never be greater than adjustedBalance1	✅	N/A	❌	10M+
GK-04	When buying tokens outside of the snipe protection period amountOutReceived == uniswapAmountOut (amount1Out)	✅	✅	✅	10M+
GK-05	Selling g8keepTokens should deduct the correct number of tokens from sender balance	✅	✅	✅	10M+
GK-06	When selling g8keepTokens amountOutReceived ==~ uniswapAmountOut (amount1Out)	✅	✅	✅	10M+
GK-07	Adding liquidity must deduct correct number of asset 0 tokens	✅	✅	✅	10M+
GK-08	Adding liquidity must deduct correct number of asset 1 tokens	✅	✅	✅	10M+
GK-09	Adding liquidity must credit recipient lp tokens calculated including fees taken out for g8keep	✅	✅	✅	10M+
GK-10	Removing liquidity must deduct the correct number of UniswapV2Pair assets from sender	✅	✅	✅	10M+
GK-11	Removing liquidity must credit the correct number of paired tokens to recipient	✅	✅	✅	10M+
GK-12	Removing liquidity must credit the correct number of g8keepTokens to recipient minus fees	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
GK-13	g8keepToken.approve() must set allowance of spender for owner to amount	✅	✅	✅	10M+
GK-14	g8keepToken.transfer() must not cause sender balance to underflow	✅	✅	✅	10M+
GK-15	g8keepToken.transfer() must deduct the correct number of tokens from sender	✅	✅	✅	10M+
GK-16	g8keepToken.transfer() must not cause recipient balance to overflow	✅	✅	✅	10M+
GK-17	g8keepToken.transfer() must credit the correct number of tokens to recipient	✅	✅	✅	10M+
GK-18	g8keepToken.transfer() to self must not affect sender balance on self transfer	✅	✅	✅	10M+
GK-19	g8keepToken.transfer() to self must not affect recipient balance on self transfer	✅	✅	✅	10M+
GK-20	g8keepToken.transferFrom() must not cause sender balance to underflow	✅	✅	✅	10M+
GK-21	g8keepToken.transferFrom() must deduct the correct number of tokens from sender	✅	✅	✅	10M+
GK-22	g8keepToken.transferFrom() must not cause recipient balance to overflow	✅	✅	✅	10M+
GK-23	g8keepToken.transferFrom() must credit the correct number of tokens to recipient	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
GK-24	g8keepToken.transferFrom() to self must not affect sender balance on self transfer	✅	✅	✅	10M+
GK-25	g8keepToken.transferFrom() to self must not affect recipient balance on self transfer	✅	✅	✅	10M+
GK-26	If sender allowance is not type(uint256).max, g8keepToken.transferFrom() must not cause sender allowance to underflow	✅	✅	✅	10M+
GK-27	If sender allowance is not type(uint256).max, g8keepToken.transferFrom() must deduct allowance from sender	✅	✅	✅	10M+
GK-28	g8keepVester.claim() must credit the correct number of vested tokens to vesting recipient	✅	✅	✅	10M+
GK-29	vested() token amount is always <= contract balance	✅	❌	✅	10M+
GK-30	vested() amount is always less than or equal to total tokens vested	✅	❌	✅	10M+
GK-31	totalSupply of g8keep should be equal to sum of balances	✅	✅	✅	10M+
GK-32	Vesting end should should always be greater than start	✅	❌	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Vested Calculations Are Broken	DoS/Gaming	Critical	Resolved
H-01	DoS for LPs Removing Liquidity	Logical Error	High	Acknowledged
H-02	Reduced Penalty When Trading In Batches	Gaming	High	Acknowledged
H-03	Deployers Can Escape From Initial Liquidity Fee	Gaming	High	Resolved
M-01	Missing Validation For _initialLiquidity	Validation	Medium	Resolved
M-02	Liquidity Providers Are Charged Trading Fees	Logical Error	Medium	Acknowledged
M-03	Excessive Trade Penalty Charged	Logical Error	Medium	Resolved
M-04	Flash Swaps Unavailable During Snipe Protection	Logical Error	Medium	Acknowledged
M-05	Swap Fees Values Not Supported	Logical Error	Medium	Resolved
M-06	All Token Deployments Can Be DoS'ed	DoS/Gaming	Medium	Resolved
M-07	Malicious Code Can Be Set In Token Name / Symbol	XSS	Medium	Resolved
M-08	Gatekeep May Avoid Buy Taxes	Gaming	Medium	Resolved
M-09	Minimum Token Balance Can Be Inaccurate	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-10	Lacking SafeCast Usage	Best practice	Medium	Resolved
L-01	Tokens Remain Max Approved	Logical Error	Low	Resolved
L-02	Unexpected Claims For Token Deployer	Access Control	Low	Resolved
L-03	Unused Code	Optimization	Low	Resolved
L-04	Users Can't Deploy Tokens With WETH	Logical Error	Low	Resolved
L-05	Avoid Using block.timestamp For Swaps	Logical Error	Low	Resolved
L-06	Avoid Dumping Token Fees During Snipe Protection	Logical Error	Low	Resolved
L-07	Paired Token Should Use SafeTransferLib	Best practice	Low	Resolved
L-08	Deployer Fees Can Be Burned	Validation	Low	Resolved
L-09	Lack Of Events Emitted	Best practice	Low	Resolved
L-10	Router Returns Higher Amounts	Logical Error	Low	Acknowledged
L-11	Penalty Logic Should Be Documented	Documentation	Low	Acknowledged
L-12	Redundant Timestamp Check In Vesting	Superfluous Code	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-13	Redundant Withdraw Function	Superfluous Code	Low	Resolved
L-14	Revert On Zero Transfer Tokens Can Cause DoS	Non-Standard Tokens	Low	Resolved
L-15	Unused Custom Error	Superfluous Code	Low	Resolved
L-16	Unexpected Transfer Amount Emitted	Best practice	Low	Acknowledged
L-17	Max Balance Transfer Tokens Behavior	Non-Standard Tokens	Low	Resolved
L-18	Lacking Approval Event	Best practice	Low	Resolved
L-19	Misleading Swap Events	Best practice	Low	Resolved
L-20	Lacking Vester Access Control	Access Control	Low	Acknowledged
L-21	Unrestricted maximumSnipeProtectionSeconds	Validation	Low	Resolved
L-22	maxSnipeProtectionBuyWithoutPenalty Inconsistency	Validation	Low	Resolved

C-01 | Vested Calculations Are Broken

Category	Severity	Location	Status
DoS/Gaming	● Critical	g8keepVester.sol: 99	Resolved

Description PoC

Users can deploy a token and select a portion of the initial supply to vest over a period of time. The issue arises when calculating the vested amount, specifically in `timeSinceLastClaim`, as the number will be huge due to the fact that `vesting.lastClaim` is never initialized.

There are two main impacts with this issue:

1. Deployer will be DoS'ed when trying to claim tokens, as the vested amount calculation will be much higher than the contract balance. Consider this scenario:
• block.timestamp = 1720483200
• vestingPeriod = 7 days
• vestingAmount = 1000e18
• vestedAmount = 1000e18 * 1720483200 / (7*24*60*60) ~= 2844714e18

2. A malicious deployer can set a vest time equal to the current `block.timestamp` and immediately claim all tokens after deployment. Therefore, deployer can trick the token holders to think the vesting time is huge (i.e. 55 years) but deployer can claim them at any time and sell at market price, rug pulling all users. Consider this scenario:

• block.timestamp = 1720483200
• vestingPeriod = 1720483200
• vestingAmount = 1000e18
• vestedAmount = 1000e18 * 1720483200 / 1720483200 = 1000e18

Recommendation

Initialize the `lastClaim` param when deploying the vest:
`deploymentVesting.lastClaim = uint40(block.timestamp);`

Resolution

G8Keep Team: Resolved.

H-01 | DoS for LPs Removing Liquidity

Category	Severity	Location	Status
Logical Error	High	g8keepToken.sol: 269	Acknowledged

Description

During the snipe protection phase, a user who decides to add liquidity to the uniswap pool will be prevented from removing the liquidity. Their tokens will remain locked in the pool until `_adjustAmountOut()` will no longer be called in the `_transfer` flow.

This occurs because at the time `_adjustAmountOut()` is called, the tokens for `token0` have already been transferred to the user. Which, in turn, will cause `balance0` to be less than `reserve0`, triggering an `InsufficientPoolInput` revert.

Recommendation

Instead of reverting when this occurs, return the value of `amount1Out` so that users can LP to the pool without facing price impact.

Resolution

G8Keep Team: Should document removing liquidity is not allowed until after the end of snipe protection.

H-02 | Reduced Penalty When Trading In Batches

Category	Severity	Location	Status
Gaming	High	g8keepToken.sol: 287	Acknowledged

Description PoC

The protocol states Purchases that would decrease the balance below the expected balance are penalized exponentially by reducing the output amount.

The issue relies on the adjustedAmount1Out calculation when applying penalty, as users can opt to do multiple smaller swaps instead of a big swap to reduce the penalty imposed.

Recommendation

Consider making the snipe protection penalty linear so it cannot be gamed by batch buys. Otherwise to maintain the penalty's exponential nature consider basing the exponential cost on the aggregate buys that have exceeded the expected amount in the current block rather than the current buy.

Or base this penalty on the amount of buys within a given lookback period. This way malicious actors cannot game the penalty with multiple buys in a single transaction and therefore must risk spreading their buys out over multiple blocks.

Finally if this behavior is acceptable, be aware of this gaming mechanism and warn users.

Resolution

G8Keep Team: Acknowledge and document.

H-03 | Deployers Can Escape From Initial Liquidity Fee

Category	Severity	Location	Status
Gaming	High	g8keepFactory.sol: 66	Resolved

Description PoC

Users can permissionlessly deploy g8keepTokens and they have to pay 2% initial liquidity fee for deployment. All of the initial liquidity except this fee, and all of the total supply except deployer's share will be added to liquidity pool during deployment.

During the `addLiquidity`, Uniswap router transfers tokens from `g8keepFactory` to Uniswap pair contract, and liquidity amount is calculated based on previous reserves and added token balances. Since this function is called immediately after deployment, reserves are 0 for both tokens and total LP supply is also 0. However, balances of the pair contract can be manipulated before its deployment.

Consider a scenario where a deployer wants to deploy 100 WETH / 10_000_000 g8Token pair. If the user deploys this as expected, they will have to pay 2 WETH initial liquidity fee. However, they can transfer ~98 WETH directly to the pair by precomputing the pair address, and deploy with 0.1 WETH / 10_000_000 g8Token parameters using the `g8keepFactory`. This way, they will only pay 0.002 WETH fee while deploying the exact same amount of tokens.

Additionally, deployers may avoid paying the liquidity fee to gatekeep by first deploying a small amount of paired token liquidity and then adding more paired tokens directly after launch and triggering a sync in Uniswap V2.

Recommendation

Check the `pairedToken` balance of the newly deployed pair contract before adding the initial liquidity, and revert if a user preemptively added some tokens to this address. Additionally consider the case where users add paired tokens after the initial deployment and set the minimum paired liquidity requirement accordingly.

Resolution

G8Keep Team: We skim the pool for any value that is deposited pre-deployment, add that amount to the initial liquidity and apply the g8keep liquidity fee to the full amount.

M-01 | Missing Validation For _initialLiquidity

Category	Severity	Location	Status
Validation	Medium	g8keepFactory.sol: 91	Resolved

Description PoC

Users that choose WETH as the paired token will need to send the `_initialLiquidity` as ETH using `msg.value`. Therefore, all ETH sent will be converted to WETH, but any excess ETH sent above `_initialLiquidity` value will not be used for adding liquidity to the pool, and left stuck in the contract.

The `g8keepFactory` will need to use `withdrawToken` to recover these stuck WETH funds from users. If an attacker realized the Factory contains WETH tokens, he can trigger a new token deployment, and use the WETH as part of their own liquidity, and send less or no ETH with msg.value.

Recommendation

Consider validating if `msg.value == _initialLiquidity` when `_pairedToken == WETH`.

Resolution

G8Keep Team: Resolved.

M-02 | Liquidity Providers Are Charged Trading Fees

Category	Severity	Location	Status
Logical Error	Medium	g8keepToken.sol: 206	Acknowledged

Description

The g8Token `_transfer` function includes a fee on transfer logic, which means users incur sell fees when `to == UNISWAP_V2_PAIR` and buy fees when `from == UNISWAP_V2_PAIR`.
The problem with this logic is that liquidity providers will face fees when adding or removing liquidity, contrary to the intended protocol design.

Recommendation

As it is not straightforward to distinguish between a liquidity action and a swap during the `_transfer` execution, we suggest documenting this behavior. This way, liquidity providers can be informed about the buy/sell fee.

Resolution

G8Keep Team: Acknowledged.

M-03 | Excessive Trade Penalty Charged

Category	Severity	Location	Status
Logical Error	Medium	g8keepToken.sol: 276	Resolved

Description

A penalty is only supposed to be applied to a trade that drops the amount of gatekeep tokens to below the expected balance. Since 996 is used instead of 997 in the calculation of `minimumToken1Balance`, a user will be charged an additional 1/3 of the Uniswap fee when the expected balance is in surplus.

Additionally, this creates a divergence between the reported maximum buy that would not be penalized from the `maxSnipeProtectionBuyWithoutPenalty` function which will be displayed on the frontend and the amount that is actually allowed without penalization upon a transfer from the Uniswap pair.

Recommendation

The calculation of `minimumToken1Balance` aims to determine the amount of token1s required to remain in the pair to satisfy `x*y=k`. Uniswap already performs the calculations necessary to achieve this without worrying about `x*y=k` being thrown out of sync.

Simply set `minimumToken1Balance` to `balance1 - amount1Out`. `balance1` is used instead of `reserve1` to stay in-line with the `maxSnipeProtectionBuyWithoutPenalty` functionality, which also relies on the balance rather than the reserve. Then you can set the initial value of `adjustedAmount1Out` to `amount1Out`. This will make the logic more accurate and reduce gas costs.

Resolution

G8Keep Team: Resolved.

M-04 | Flash Swaps Unavailable During Snipe Protection

Category	Severity	Location	Status
Logical Error	Medium	g8keepToken.sol: 276	Acknowledged

Description

Flash swaps is a core feature of Uniswap V2. Users are able to use the `swap` function to obtain tokens in the pool, and use the `uniswapCall` callback to repay the tokens. The issue arises when users try to flash swap the g8keep token during sniping protection window, as the `minimumToken1Balance` will be the same as the `balance1`, so `adjustedAmount1Out` will be 0.

Therefore, the flash swap feature will not transfer tokens to the user during sniping protection window. Additionally, after the window is over, flash swaps are enabled, as `_adjustAmountOut` is not invoked, but users will be charged both a buy fee to take the flash loan and sell fee during repayment.

Recommendation

Document this behavior so users are aware that flash swaps are disabled during sniping protection.

Resolution

G8Keep Team: Acknowledged.

M-05 | Swap Fees Values Not Supported

Category	Severity	Location	Status
Logical Error	Medium	g8keepFactory.sol: 71	Resolved

Description

Users can deploy tokens with specific buy and sell fees. These values are passed as params in the `g8keepFactory` constructor. The issue is that both `_buyFee` and `_sellFee` are `uint8`, so it won't support any values above 255, although the max value for both fees is set to 500.

Recommendation

Consider updating the `_buyFee` and `_sellFee` value type to `uint16` to support higher fee values.

Resolution

G8Keep Team: Resolved.

M-06 | All Token Deployments Can Be DoS'ed

Category	Severity	Location	Status
DoS/Gaming	Medium	g8keepToken.sol: 93	Resolved

Description PoC

Users can permissionlessly create new g8keepTokens via the factory contract. This process includes deployment of the new token, creation of a pair in Uniswap with this new token, and adding liquidity to the Uniswap pair. All of these actions happen in a single transaction.

New Uniswap pair creation is done in the g8keepToken constructor with this line: UNISWAP_V2_PAIR = IUniswapV2Factory(uniswapV2Router.factory()).createPair(address(this), _pairedToken)

The createPair function in Uniswap factory:
1. Reverts when there is already a pair with same token addresses.
2. Does not check whether inputted token addresses are actually exist or not.
3. Can be called by anyone.

Lastly, g8keepFactory contract uses create2 method while deploying g8keepTokens, which means anyone can precompute the future g8keepToken address. An attacker can precompute the new token address and directly call the createPair function in Uniswap factory before it is deployed, causing the deployment to fail.

Recommendation

Consider performing an action similar to _addLiquidity function in UniswapRouter (Source), and check if the pair already exists instead of always calling createPair in the constructor.

Resolution

G8Keep Team: Resolved with try/catch for pair creation.

M-07 | Malicious Code Can Be Set In Token Name / Symbol

Category	Severity	Location	Status
XSS	Medium	g8keepFactory.sol: 67	Resolved

Description

The g8keepFactory allows users to deploy a custom g8keepToken including the ability to set a name and symbol for the new token. However, it is possible for an attacker to craft a name or symbol such that it includes markup that can contain Javascript code.

If loaded into a frontend without XSS protection, this can cause potential harm to users of the platform as was the case in the EtherDelta exploit. For more details on the EtherDelta exploit please refer to this article.

Recommendation

Sanitize or limit the length of the token `_name` and `_symbol` passed into the `deployToken` function from the `g8keepFactory` contract.

Resolution

G8Keep Team: Addressed in g8keep UI/backend integration.

M-08 | Gatekeep May Avoid Buy Taxes

Category	Severity	Location	Status
Gaming	Medium	g8keepToken.sol: 206	Resolved

Description

In the `_transfer` function for the `g8keepToken` contract the buy taxes implemented by the token deployer are ignored if the receiver of the buy is the gatekeep factory contract.
However under normal operation there is no use-case for the gatekeep factory to be the receiver of a buy.

This behavior allows the gatekeep owner to buy deployed g8keep tokens without the buy tax. These tokens may be withdrawn with the `withdrawToken` function or sold with the `sellTokens` function.
As this is functionality is not planned to be an explicit feature of the system, the admin should not be able to circumvent the buy fees.

Recommendation

Consider removing the `!to == G8KEEP` condition in the _transfer function so that buy fees cannot be avoided by the g8keep owner.

Resolution

G8Keep Team: Resolved.

M-09 | Minimum Token Balance Can Be Inaccurate

Category	Severity	Location	Status
Logical Error	Medium	g8keepToken.sol: 273	Resolved

Description

In the `_adjustAmountOut` function the amount0In amount computed by the `g8keepToken` contract is based upon the `balance0 - reserve0`. However in the UniswapV2 pair contract a user may specify a nonzero amount0Out and a nonzero amount1Out.

In this case the `amount0In` in the `swap` function is computed as `balance0 - (_reserve0 - amount0Out)` which does not match the `amount0In` computation in the g8keepToken `_adjustAmountOut` function.

As a result the `amount0In` value in the `_adjustAmountOut` function is smaller in these cases, causing the `minimumToken1Balance` to be smaller and thus the `adjustedAmount1Out` to be larger than it should be.

Recommendation

Consider implementing the simplification mentioned in M-03 which avoids this accounting difference. Otherwise be aware of this difference in the swap fees computed by the Uniswap V2 pair and the swap fees computed in the `_adjustAmountOut` function and document it's effect on the snipe protection penalty.

Resolution

G8Keep Team: M-03 recommendations implemented.

M-10 | Lacking SafeCast Usage

Category	Severity	Location	Status
Best practice	Medium	Global	Resolved

Description

Throughout the g8keep codebase raw casts are made which potentially dangerously downcast values that may overflow the uint sizes they are casted into.

One instance in the `deploymentVest` function invalidates the invariant `GK-32`, `"Vesting end should always be greater than the vesting start"` as the `block.timestamp + _vestTime` can exceed the maximum `uint40` value when the provided `_vestTime` is exceptionally large.

Recommendation

Implement SafeCast throughout the codebase to avoid all potential overflow issues. Otherwise carefully examine and implement the appropriate validations for all cases where values are downcast.

Resolution

G8Keep Team: Resolved with check on vestingEnd to not be greater than type(uint40).max, other casts such as balance/reserve to uint112/uint128 are safe as total supply is checked to not exceed type(uint40).max, added constant for max setting of max snipe protection seconds.

L-01 | Tokens Remain Max Approved

Category	Severity	Location	Status
Logical Error	Low	g8keepFactory.sol: 144	Resolved

Description

When removing a token from `allowedPairs` using `setPairedTokenSettings()`, the token is set to max approval again. This means once a token is given max approval, the token will always have max approval.

Recommendation

If `allowed` is false, set the token approval to 0.

Resolution

G8Keep Team: Resolved. Added a function setApprovalToUniswapRouter that performs a similar function to the setPairedTokenSettings without adding it to allowed pairs, this serves to allow the sellTokens function to be utilized in the event that a token approval is accidentally revoked through setPairedTokenSettings without having to temporarily allow it to be a paired token.

L-02 | Unexpected Claims For Token Deployer

Category	Severity	Location	Status
Access Control	Low	g8keepVester.sol: 66	Resolved

Description

The `claim` function does not have any access control, therefore any user can claim on behalf of the deployer. Although deployer will be the recipient of the tokens, he might not want to claim them yet and leave tokens in the Vester contract. Might want to wait the whole vesting period to show the commitment to the project, as claiming them can suggest he will sell and dump the price

Recommendation

Consider adding an access control, to only allow vesting recipient to claim tokens.

Resolution

G8Keep Team: Resolved.

L-03 | Unused Code

Category	Severity	Location	Status
Optimization	Low	g8keepToken.sol, g8keepFactory.sol	Resolved

Description

The following functions have internal visibility, but they are never used:
• _getToken0Reserves
https://github.com/GuardianAudits/gatekeep-1/blob/main/src/g8keepToken.sol#L252
• _getToken1Reserves
https://github.com/GuardianAudits/gatekeep-1/blob/main/src/g8keepToken.sol#L256

Additionally, the FeeTransferFailed in the Factory contract is never used:
https://github.com/GuardianAudits/gatekeep-1/blob/main/src/g8keepFactory.sol#L37C11-L37C28

Recommendation

Remove the unused code from the contracts.

Resolution

G8Keep Team: Removed the unused error, _getToken0Reserves and _getToken1Reserves are now used in different functions and _getTokenReserves is unused/deleted.

L-04 | Users Can't Deploy Tokens With WETH

Category	Severity	Location	Status
Logical Error	Low	g8keepFactory.sol: 90	Resolved

Description

Users will use WETH as the main paired token when deploying `g8keepTokens`. The contract will wrap the ETH send in `msg.value` to obtain WETH tokens. If users have WETH token balance and not ether, they will need to unwrap them first and then deploy the g8keepToken.

Recommendation

Consider refactoring the `_pairedToken` and `msg.value` checks, to allow users to transfer in WETH tokens when `_pairedToken==WETH` and `msg.value == 0`.

Resolution

G8Keep Team: Resolved.

L-05 | Avoid Using block.timestamp For Swaps

Category	Severity	Location	Status
Logical Error	Low	g8keepFactory.sol: 216	Resolved

Description

The factory contract will receive g8keepToken fees for swaps. Owner will then use the `sellTokens` admin function to sell these fees for the paired token.

The issue arises when using `block.timestamp` as the deadline parameter for `swapExactTokensForTokensSupportingFeeOnTransferTokens`. A malicious block builder will be able to execute this at any time, when such transaction is useful for manipulating the price.

Recommendation

Add a `deadline` parameter to the `sellTokens` and use this instead of `block.timestamp` for all the swaps.

Resolution

G8Keep Team: Resolved.

L-06 | Avoid Dumping Token Fees During Snipe Protection

Category	Severity	Location	Status
Logical Error	Low	g8keepFactory.sol: 203	Resolved

Description

The owner of `g8keepFactory` has the ability to execute `sellTokens` in order to unload all fees acquired from g8token exchanges, without any sell fees being charged, which is in line with expectations. However, when snipe protection is activated, selling off the tokens not only decreases the token price, but also increases the `token1` reserves, hindering the proper application of penalties.

Recommendation

It is advised to restrict the owner from executing `sellTokens` when the g8keepToken has active snipe protection.

Resolution

G8Keep Team: Resolved.

L-07 | Paired Token Should Use SafeTransferLib

Category	Severity	Location	Status
Best practice	Low	g8keepFactory.sol	Resolved

Description

Any paired token can be added by the owner. In order to maintain compatibility with as many tokens as possible, `safeTransfer` and `safeTransferFrom` ought to be used to validate returned values.

Recommendation

Use `safeTransfer` and `safeTransferFrom` from the solady `SafeTransferLib` when transferring the paired token.

Resolution

G8Keep Team: Resolved.

L-08 | Deployer Fees Can Be Burned

Category	Severity	Location	Status
Validation	Low	g8keepToken.sol: 227	Resolved

Description

The `treasuryWallet` receives buy and sell fees in g8Token, during `_applyFees` execution.
The issue is that this address is not validated neither in the Factory contract or token deployment, like it's done in the `updateTreasuryWallet` owner function.

Therefore, `address(0)` is be a valid value, so every buy and sell fee tokens will be burned, and emit a `Transfer` event with `to` address equal to `address(0)`. Burning tokens should reduce the `totalSupply` but this is an immutable variable, as minting and burning should not be allowed.

Recommendation

Validate if `_treasuryWallet` is not `address(0)` during token deployment.

Resolution

G8Keep Team: Resolved.

L-09 | Lack Of Events Emitted

Category	Severity	Location	Status
Best practice	Low	Global	Resolved

Description

The following main functions lack event emissions:

g8keepFactory
• setDeploymentSettings
• setPairedTokenSettings

g8keepVester
• claim

g8keepToken
• updateTreasuryWallet

Emitting events will facilitate state changes tracking by off chain services.

Recommendation

Consider emitting events from the above functions.

Resolution

G8Keep Team: Resolved.

L-10 | Router Returns Higher Amounts

Category	Severity	Location	Status
Logical Error	Low	UniswapRouterV2.sol	Acknowledged

Description

The `UniswapRouterV2` includes a `swapTokensForExactTokens` function.
When a user directly interacts with the router contract and utilizes this function to swap WETH for tokens, the `amountOut` returned is higher than the actual amount received by the user. Additionally, when swapping tokens for WETH, the transaction will revert.

Recommendation

It is advised to document this behavior to ensure that protocols integrating `swapTokensForExactTokens` are informed about the accurate amount swapped using the router contract.

Resolution

G8Keep Team: Acknowledge and document.

L-11 | Penalty Logic Should Be Documented

Category	Severity	Location	Status
Documentation	Low	g8keepToken.sol	Acknowledged

Description

The protocol implements a snipe protection logic to prevent huge purchases. Users can buy g8keepTokens up to a point without penalty but they have to pay a penalty after that point. Penalty calculation is done by adjusting the output amount of the g8keepToken. If the remaining balance in the Uniswap enters to penalty zone, output adjustment will be performed.

According to docs: “any amount of balance over the expected balance may be purchased from the pool with zero penalty. Purchases that would decrease the balance below the expected balance are penalized exponentially by reducing the output amount”.

However, penalty is applied to the entire output amount of the trade, not just the amount that would cause the balance to go below expected balance. In some scenarios, resulted amount may become even less than the no penalty amount due to exponential penalty, and might brake “any amount of balance over the expected balance may be purchased from the pool with zero penalty” statement.

Also, since the intention is penalizing the whole trade amount, users can game the penalty logic by buying up to the max limit without penalty first, and then buying the remaining part in a second transaction to pay less penalty.

Recommendation

Consider documenting this behaviour for the users to prevent misunderstandings and possible loss of funds.

Resolution

G8Keep Team: Acknowledge and document.

L-12 | Redundant Timestamp Check In Vesting

Category	Severity	Location	Status
Superfluous Code	Low	g8keepVester.sol: 92	Resolved

Description

The `g8keepVester` contract calculates the `vestedAmount` in the internal `_vested` function which is used in the `claim` function as the amount of vested tokens to send to the deployer. There is a check if the `block.timestamp` is less than the `vestingStart` then the `vestedAmount` returned should be zero.

This suggests that the vesting can start at a future date. However, this is a redundant check since the `vestingStart` is always initialized to the `block.timestamp` such that it will always be in the past.

Recommendation

The check in L69 in the `claim` function and on L92 in the `_vested` function are not required and can be removed.

Resolution

G8Keep Team: Resolved.

L-13 | Redundant Withdraw Function

Category	Severity	Location	Status
Superfluous Code	Low	g8keepToken.sol: 176	Resolved

Description

The g8keepToken has a `withdrawETH` function to withdraw ETH balances from the contract. However, this contract does not have a receive function or any payable functions, meaning it cannot hold ETH balances. As a result, the `withdrawETH` function is considered redundant.

Recommendation

Consider removing redundant function.

Resolution

G8Keep Team: Resolved.

L-14 | Revert On Zero Transfer Tokens Can Cause DoS

Category	Severity	Location	Status
Non-Standard Tokens	Low	g8keepFactory.sol: 119	Resolved

Description PoC

The g8KeepAdmin can add new ERC20 tokens to the allowedPairs whitelist. This allows the whitelisted token to be used as liquidity in the UniswapV2 pool. Moreover, g8KeepAdmin can adjust the fee that is used for deployments so that the GateKeep protocol can run promotions.

However, if the allowedPairs includes a token that reverts on zero transfers and the g8keepInitialLiquidityFee is set to zero during a promotion, then the deployToken function in the g8keepFactory contract will revert.

This is because on L119 in the g8keepFactory contract the fee to transfer to the g8keepFeeWallet will be zero and given the _pairedToken in this scenario will revert on zero transfers this will prevent users from deploying a new g8KeepToken.

This is a very specific scenario where the user wants to deploy a new g8KeepToken paired with a token that reverts on zero transfers during a promotion when the fee is set to zero.

Recommendation

Include a check that the g8keepInitialLiquidityFee is greater than zero before performing the fee transfer to the g8keepFeeWallet.

Resolution

G8Keep Team: Resolved.

L-15 | Unused Custom Error

Category	Severity	Location	Status
Superfluous Code	Low	g8keepVester.sol: 40	Resolved

Description

The custom errors `FeeTransferFailed` in `g8keepFactory` contract and `PoolReservesNotSynced` in `g8keepToken` contract are defined but are never used.

Recommendation

Remove unused errors.

Resolution

G8Keep Team: Resolved.

L-16 | Unexpected Transfer Amount Emitted

Category	Severity	Location	Status
Best practice	Low	g8keepToken.sol: 216	Acknowledged

Description

In the transfer function the `Transfer` event emits the toAmount which has had any buy or sell fees applied. As a result the Transfer event will emit the received amount which does not include these fees.

This may be unexpected for consumers of the Transfer event which would attempt to track the amount which was removed from the from address.

Recommendation

Keep this potentially unexpected behavior in mind and consider documenting this for integrators.

Resolution

G8Keep Team: Transfer amounts from the from address to the fee recipients are emitted in the _applyFees function that should appropriately balance the total amount deducted from from and sent to other addresses.

L-17 | Max Balance Transfer Tokens Behavior

Category	Severity	Location	Status
Non-Standard Tokens	Low	g8keepFactory.sol: 96	Resolved

Description

Some ERC20 tokens such as cUSDCv3 have the behavior that transferring the maximum value transfers the entire account balance. Though in most cases this will result in a revert when calling the `deployToken` function and specifying an `_initialLiquidity` of `type(uint256).max`, there may be some edge cases depending on the `g8keepInitialLiquidityFee` assignment that would allows this unexpected token behavior to be used.

In such a case the caller may be able to circumvent the `pairedTokenMinimumLiquidity` value or utilize paired tokens that were held in the g8keepFactory contract.

Recommendation

Consider re-assigning the `_initialLiquidity` amount to the amount that is received after transferring in the `_pairedToken` amount. Consequently this will also fix any mis-accounting for fee-on-transfer or rebase tokens.

Resolution

G8Keep Team: Resolved.

L-18 | Lacking Approval Event

Category	Severity	Location	Status
Best practice	Low	g8keepToken.sol: 102	Resolved

Description

In the constructor for the `g8keepToken` contract the `_allowances` mapping is directly written to for the `g8keepFactory`, however no `Approval` event is emitted.

Recommendation

Add the following event emission to the constructor:
`emit Approval(msg.sender, _uniswapV2Router, type(uint256).max);`.

Resolution

G8Keep Team: Resolved.

L-19 | Misleading Swap Events

Category	Severity	Location	Status
Best practice	Low	g8KeepToken.sol	Resolved

Description

During the snipe protection window the `amount1Out` is adjusted to be the maximum amount extractable via the calculated `minimumToken1Balance`. However the Uniswap V2 pair contract will emit the `amount1Out` that was originally specified by the user which is not accurate to the adjusted amount which was sent.

For example, if the user sends 10 WETH and can receive up to 100 g8 tokens out of the swap, but only specifies an amountOut of 1 wei, the `_adjustAmountOut` function will adjust the amount they receive to be the maximum 100 g8 tokens, but the Swap event emitted by the Uniswap V2 pair will emit the original 1 wei amount.

Additionally the `amount1Out` emitted by the Uniswap pair contract will not include any buy or taxes which may be further misleading.

Recommendation

Consider implementing the recommended simplifications from M-03 to use the amount1Out as the adjustedAmount basis.

Additionally be aware that the amount emitted in the Uniswap Swap event does not include buy taxes and consider documenting this where appropriate.

Resolution

G8Keep Team: M-03 recommendations implemented.

L-20 | Lacking Vester Access Control

Category	Severity	Location	Status
Access Control	Low	g8keepVester.sol: 34	Acknowledged

Description

The `deploymentVest` function is an external function with no deliberate access control.
As a result arbitrary addresses may call the `deploymentVest` function and create invalid vests for tokens which are either invalid tokens or tokens which are not supported gatekeep tokens.

Depending on the frontend and off-chain systems this may have an effect as the `DeploymentVestCreated` event will be emitted for an invalid token.

Recommendation

Consider implementing some form of access control which would verify that only valid g8keep tokens which have been created through the official g8keepFactory contract can call the `deploymentVest` function. Otherwise be sure that these invalid event emissions will not effect the frontend or other off-chain systems.

Resolution

G8Keep Team: Acknowledge and implement controls in g8keep backend.

L-21 | Unrestricted maximumSnipeProtectionSeconds

Category	Severity	Location	Status
Validation	Low	g8keepFactory.sol: 147	Resolved

Description

In the `setDeploymentSettings` function there is no maximum threshold which the `_maximumSnipeProtectionSeconds` value is validated against. As a result the g8keep owner may configure a high `maximumSnipeProtectionSeconds` which can be problematic as described in H-01 where LP funds are locked until the snipe protection window is over.

Recommendation

Consider implementing a maximum threshold for the `maximumSnipeProtectionSeconds` configuration. Otherwise carefully consider the issue described in H-01 when assigning this value.

Resolution

G8Keep Team: Resolved.

L-22 | maxSnipeProtectionBuyWithoutPenalty Inconsistency

Category	Severity	Location	Status
Validation	Low	g8keepToken.sol: 120	Resolved

Description

The `maxSnipeProtectionBuyWithoutPenalty` view function uses the `token1` balance of the Uniswap pair contract to determine the maximum buy that will not experience a snipe protection penalty. However the actual snipe protection penalty which is applied in the `_adjustAmountOut` function relies on the reserves of the Uniswap V2 pair.

As a result in cases where there are token1s in excess of the Uniswap V2 pair reserves the `maxSnipeProtectionBuyWithoutPenalty` inaccurately reports a buy size that is larger than an amount which will go without penalty. Users who use the inaccurate `maxSnipeProtectionBuyWithoutPenalty` will be unexpectedly penalized for their entire buy amount.

Recommendation

Modify the `_adjustAmountOut` function such that the `minimumToken1Balance` relies on the token1 balance similar to the `maxSnipeProtectionBuyWithoutPenalty` function. This has been implemented in the M-03 recommendation.

Resolution

G8Keep Team: M-03 recommendations implemented.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits