PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Excalibur

02 February 2022

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 Excalibur Exchange 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	Excalibur Exchange
URL	https://excalibur.exchange/
Platform	Fantom Opera
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
GRAILToken	0x1a8fd04b7eBEB82605829EEA91324D81b41d6dcf	MATCH
EXCToken	0x6e99e0676A90b2a5a722C44109db22220382cc9F	MATCH
Dividends	0xa561f3A8e4C32b220B4C92e0e7ee5fFD2459B46B	MATCH
MasterChef	MasterChef.sol	UNDEPLOYED
MasterExcalibur	0x70B9611f3cd33e686ee7535927cE420C2A111005	MATCH
FeeManager	0xcA1D2F16b9969a393FF1Dc92704cB8A552268FBC	MATCH
ERC20BurnSupply	Helper Library	MATCH
ERC20AvgReceiveTime	Helper Library	MATCH
WrapERC20WithPenalty	Helper Library	MATCH
Multicall	0x0b67Af0F3b4028a354EEb6a88b0b051dd1Fd6D09	MATCH
ExcaliburV2Factory	0x08b3CCa975a82cFA6f912E0eeDdE53A629770D3f	MATCH
UniswapV2Pair	Deployed by ExcaliburV2Factory	MATCH
Math, SafeMath, UQ112x112	Helper Libraries	MATCH
ExcaliburRouter	0xc8Fe105cEB91e485fb0AC338F2994Ea655C78691	MATCH
PriceConsumerV3	0xF5580BE00EEfA89125308293B3BeBBbd1975A717	MATCH
UniswapV2Library	Helper Library	MATCH

[Update April 4 2022] The client has upgraded their previous deployment (`0x3f019f17c5b4Fd86D18dD59D03Dcb602a72986Cb`) to an updated version (`0xcA1D2F16b9969a393FF1Dc92704cB8A552268FBC`) with several minor changes. Paladin has carefully reviewed the latest update and only informational findings were raised, all these findings are resolved in the deployed version.

The main changes compared to the previous function are the addition of three privileged functions: `burnToken`, `burnTokens` and `updateAutoburnedToken`. Within the latest version of the FeeManager, several tokens can now be burned directly from the FeeManager once they are configured to do so.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	4	4	-	-
Medium	2	2	-	-
Low	5	4	-	1
Informational	31	30	-	1
Total	**42**	**40**	**-**	**2**

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 GRAILToken

ID	Severity	Summary	Status
01	INFO	Unused `mint` and `burn` return values	RESOLVED

1.3.2 EXCToken

ID	Severity	Summary	Status
02	INFO	Usage of `require` over `assert` when the requirement can never fail	RESOLVED

1.3.3 Dividends

ID	Severity	Summary	Status
03	MEDIUM	`updateUser` already uses the new supply for newly minted tokens	RESOLVED
04	MEDIUM	`excludeContract` does not always update the `rewardDebt` which could cause reversions in other parts of the system	RESOLVED
05	LOW	Unprivileged functions do not validate the token address	RESOLVED
06	LOW	`enableDistributedToken` does not always set the `lastUpdateTime`	ACKNOWLEDGED
07	INFO	Adding too many tokens could cause the Dividends contract to run out of gas and render the GrailToken unusable	RESOLVED
08	INFO	UI function `pendingDividendsAmount` is wrongly defined while the `nextCycle` has been exceeded but no update has occurred yet	RESOLVED
09	INFO	Gas optimization: Making `distributedTokens` an EnumerableSet would greatly simplify and optimize the code	RESOLVED
10	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
11	INFO	`grailToken` can be made immutable	RESOLVED
12	INFO	Inconsistent usage of `reentrancyGuards`	RESOLVED
13	INFO	Early return in `updateDividendsInfo` causes the "next cycle logic" to not always trigger	RESOLVED
14	INFO	Lack of SafeERC20 usage	RESOLVED

1.3.4 MasterChef

ID	Severity	Summary	Status
15	INFO	Unused variable: `lpSupply`	RESOLVED
16	INFO	`_excToken`, `_grailToken` and `startTime` can be made `immutable`	RESOLVED
17	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	RESOLVED
18	INFO	Inconsistent modifiers	RESOLVED
19	INFO	Undetermined gas usage: At some point adding pools might run out of gas	RESOLVED

1.3.5 MasterExcalibur

ID	Severity	Summary	Status
20	INFO	Users can receive a 50% bonus while being able to withdraw at any time	RESOLVED
21	INFO	Inconsistent usage of `_msgSender()`	RESOLVED
22	INFO	`pendingRewardsOnLockSlot` lacks a `validateSlot` modifier and can revert if users have a zero deposit amount	RESOLVED

1.3.6 FeeManager

ID	Severity	Summary	Status
23	HIGH	`updateShares` wrongly checks the validity of `buybackAndBurnShare_` causing it to be uncallable by governance	RESOLVED
24	INFO	Lack of `minimumReceived` parameter in swaps might allow for frontrunning	RESOLVED
25	INFO	Gas optimization: Unnecessary addition within Uniswap router operations	RESOLVED
26	INFO	`excToken`, `dividendsContract`, `safundsAddress` and `buybackAndBurnAddress` can be made `immutable`	RESOLVED

1.3.7 ERC20BurnSupply

No issues found.

1.3.8 ERC20AvgReceiveTime

No issues found.

1.3.9 WrapERC20WithPenalty

ID	Severity	Summary	Status
27	HIGH	Lack of constructor validation	RESOLVED

1.3.10 Multicall

No issues found.

1.3.11 ExcaliburV2Factory

ID	Severity	Summary	Status
28	INFO	Lack of events for `setOwner` and `setFeeToSetter`	RESOLVED

1.3.12 UniswapV2Pair

ID	Severity	Summary	Status
29	HIGH	Setting the `ownerFeeShare` to zero prevents all minting and breaking of LP tokens	RESOLVED
30	INFO	Protocol fee will be slightly less than expected	RESOLVED
31	INFO	Lack of events for `drainWrongToken`	RESOLVED
32	INFO	Gas optimization: Lack of `calldata` usage on `internal` function	RESOLVED

1.3.13 UniswapV2Erc20

ID	Severity	Summary	Status
33	INFO	Approval event is not emitted if allowance is changed in `transferFrom` as suggested in the ERC-20 Token Standard (also present in Uniswap)	RESOLVED
34	INFO	`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)	RESOLVED

1.3.14 Math, SafeMath, UQ112x112

No issues found.

1.3.15 ExcaliburRouter

ID	Severity	Summary	Status
35	HIGH	Fee rebate mechanism uses pair prices which can be easily manipulated and lacks robustness	RESOLVED
36	LOW	Typographical error: Contract defines the USD price as BUSD in the variable names	RESOLVED
37	INFO	Governance privilege: price consumer can be used or changed to potentially mint excessive excalibur tokens	ACKNOWLEDGED
38	INFO	Adding logic to the fallback function reduces the limited gas stipend of WETH withdrawals which could make them more likely to revert under protocol upgrades	RESOLVED

1.3.16 PriceConsumerV3

ID	Severity	Summary	Status
39	LOW	`_getWETHFairPriceUSD` and `_getTokenFairPriceUSD` do not revert if the price is negative or stale	RESOLVED
40	LOW	Typographical error: Contract defines the USD price as BUSD in the variable names	RESOLVED
41	INFO	`_getTokenPriceUSDUsingPair` does not properly handle decimals	RESOLVED
42	INFO	`getTokenFairPriceUSD` and `getWETHFairPriceUSD` can be made external	RESOLVED

1.3.17 UniswapV2Library

No issues found.

2 Findings

2.1 GRAILToken

The Grail token is an ERC-20 token which extends the `WrapERC20WithPenalty` dependency functionality described later on this report. It allows for GRAIL to be converted to newly minted Excalibur at a rate slightly worse than 1:1. The rate improves linearly from over a configurable period of at least 3 days and at most 14 days, which means that users will receive a better rate if they wait 3 days to convert their GRAIL to Excalibur than if they convert GRAIL immediately. For example, 1 GRAIL would be worth 0.9 Excalibur compared to 0.7 Excalibur.

Holding the GrailToken makes users eligible for dividends in various currencies which are distributed through the Dividends contract. Although distribution is automatic, the user still needs to claim these.

2.1.1 Token Overview

Address	TBC
Token Supply	TBC
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	TBC

2.1.2 Privileged Roles

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

- `initializeMasterContractAddress [ settable once ]`
- `initializeEXCConverterFactoryContractAddress [ settable once ]`
- `initializeDividendsContract [ settable once ]`
- `updateUnwrapPenaltyPeriod`
- `mint [ only by Masterchef contract ]`
- `transferOwnership`
- `renounceOwnership`

2.1.3 Issues & Recommendations

Issue #01	Unused `mint` and `burn` return values
Severity	INFORMATIONAL
Description	The `mint` and `burn` functions return boolean values which are not used throughout the system.
Recommendation	Consider removing the return values as we are not aware of any standard which requires them. This issue will also be resolved on the preference to keep these, all though we then require these to be used in the other contracts.
Resolution	RESOLVED The client has removed the `burn` return value and has indicated they would like to maintain the `mint` return value as it is used by the router, even though it always returns true.

2.2 EXCToken

The Excalibur token is a token which inherits ERC20BurnSupply. With each transfer, up to 2% (configurable by governance) is burned. During contract creation, a configurable initial supply is minted to a configurable recipient.

2.2.1 Token Overview

Address	TBC
Token Supply	TBC
Decimal Places	18
Transfer Max Size	
Transfer Min Size	
Transfer Fees	
Pre-mints	TBC

2.2.2 Privileged Roles

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

- `initializeMasterContractAddress [ callable once ]`
- `initializeDivTokenContractAddress [ callable once ]`
- `initializeRouterContractAddress [ callable once ]`
- `mint [ callable by master, divToken and router ]`
- `updateAutoBurnRate [ up to 2% ]`
- `updateExcludedFromAutoBurn`
- `transferOwnership`
- `renounceOwnership`

2.2.3 Issues & Recommendations

Issue #02	Usage of require over assert when the requirement can never fail
Severity	INFORMATIONAL
Location	Line 203 `require(amount == sendAmount + burnAmount, "EXCToken: invalid burn amount");`
Description	Requirements that can never fail can be marked as such using the keyword `assert` instead.
Recommendation	Consider using the keyword `assert` instead of `require`.
Resolution	RESOLVED

2.3 Dividends

The Dividends contract allows users to passively receive dividends in a number of configured tokens from holding the Grail Token. Dividends automatically accumulate and can be withdrawn by harvesting them. The contract owner can specify different contracts that can forward tokens to the Dividends contract as dividends. A part of these tokens are then distributed linearly over the next cycle to all token holders based on the number of tokens they hold. Each cycle, a part of the pending token rewards balance is enabled for usage in the next cycle.

The system is also not limited to one kind of token but can distribute multiple tokens as dividends at the same time.

2.3.1 Issues & Recommendations

Issue #03	`updateUser` already uses the new supply for newly minted tokens
Severity	MEDIUM SEVERITY
Description	The reward logic within the Dividends contract is wrongly defined as it uses the original user balance, but it uses the `totalSupply` of the GRAIL token after the transfer has occurred. This supply is not correct for mint and burns as it will have increased/decreased over this action. This causes the reward accounting mechanism to be defined slightly wrong over these actions.
Recommendation	Consider passing the previous `totalSupply` with `updateUser` and using this for all `activeGrailSupply` calculations.
Resolution	RESOLVED The client has followed the implementation. However, at first, the code was wrongly implemented where an `internal` function was still marked as `public`, allowing for reward inflation by a malicious party. Also, a function was being called with the wrong parameters. In the second resolution round, these bugs were patched.

Issue #04	**`excludeContract` does not always update the `rewardDebt` which could cause reversions in other parts of the system**
Severity	MEDIUM SEVERITY
Description	The `excludeContract` function only updates the `rewardDebt` of the user if their current balance is greater than zero, the same goes for `_updateUser`. This causes issues in the following sequence: 1. Exclude Alice with a balance of 1000 tokens. Alice's `rewardDebt` remains frozen at for example 5000. 2. Alice transfers out the 1000 tokens to Bob. Alice's `rewardDebt` remains frozen at 5000 but now has a balance of 0. 3. Include Alice again. 4. Alice now has a balance of 0 but a `rewardDebt` of 5000. This causes subtraction underflow reversions throughout the system
Recommendation	Consider always updating `rewardDebt`.
Resolution	RESOLVED The `rewardDebt` is now always updated in both locations.

Issue #05	Unprivileged functions do not validate the token address
Severity	LOW SEVERITY
Description	Many of the unprivileged functions do not validate that the token address provided actually exists. This is usually considered bad practice as an exploiter can provide any contract as this address, which potentially allows for privilege escalation and other side-effects. In this case, we did not find such exploits but as there is no good reason to allow users to harvest on arbitrary tokens we believe this to be an important privilege to cut down on.
Recommendation	Consider validating that any token provided by unprivileged users is included in an EnumerableSet of tokens. Using such an EnumerableSet over an array has been recommended in a later issue and serves value in this issue as well.
Resolution	RESOLVED EnumerableSet is now used and some of the functions are validated, others require the token to have existed.

Issue #06	`enableDistributedToken` does not always set the `lastUpdateTime`
Severity	LOW SEVERITY
Description	The `enableDistributedToken` function does not always set the `lastUpdateTime` to the current timestamp, this could potentially cause an issue if the token is distributed for a long time and potentially distribute rewards in hindsight over this period.
Recommendation	Consider always updating the `lastUpdateTime`. Consider adding a test case where the token is disabled for a long period (while there's still a pending distribution balance) to validate our hypothesis.
Resolution	ACKNOWLEDGED The client has indicated that this is desired behavior.

Issue #07	**Adding too many tokens could cause the Dividends contract to run out of gas and render the GrailToken unusable**
Severity	INFORMATIONAL
Description	The `updateUser` function which is called on each GrailToken transfer loops over all dividend tokens, this could become extremely costly and even render the contracts unusable if too many tokens were added.
Recommendation	Consider adding a maximum number of active dividend tokens, eg. 20. If an EnumerableSet is used, simply `require(distributedTokens.length() < 20)` before new tokens are enabled.
Resolution	RESOLVED A maximum of 10 tokens has been introduced.

Issue #08	**UI function `pendingDividendsAmount` is wrongly defined while the `nextCycle` has been exceeded but no update has occurred yet**
Severity	INFORMATIONAL
Location	Lines 186-188 `dividendAmountPerSecond_ = (dividendsInfo_.pendingAmount.mul(dividendsInfo_.cycleDividendsPercent).div(100)).div( cycleDurationSeconds );`
Description	The `dividendAmountPerSecond` should have 1e2 precision, however this precision multiplier is not included within the `pendingDividendsAmount` function. It should also be noted that the `pendingDividendsAmount` function seems to be wrongly defined if the current cycle is outdated for two cycles. As this scenario is highly unlikely, we do not expect the client to address it in a UI function.
Recommendation	Consider updating this code section as follows: `dividendAmountPerSecond_ = (dividendsInfo_.pendingAmount.mul(dividendsInfo_.cycleDividendsPercent).mul(1e2).div(100)).div( cycleDurationSeconds );` `mul(1e2).div(100)` can then be removed, though we recommend keeping a comment about this in the code to explain this.
Resolution	RESOLVED `.div(100)` has been removed.

Issue #09	**Gas optimization: Making `distributedTokens` an EnumerableSet would greatly simplify and optimize the code**
Severity	INFORMATIONAL
Description	The codebase currently uses multiple for loops that could be avoided if `distributedTokens` was an EnumerableSet (by OpenZeppelin). Such a set has a lookup and delete time complexity of O(1) which makes the code much simpler. It furthermore has an "upsert" dynamic which simply doesn't add the element twice if it already exists. `_removeTokenFromDistributedTokens` could therefore be completely deleted.
Recommendation	Consider replacing `distributedTokens` with an EnumerableSet and simplifying the code sections that use for-loops to lookup variables.
Resolution	RESOLVED EnumerableSet has been introduced.

Issue #10	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accDividendsPerShare` is based upon the `activeGrailSupply()` function. `dividendsInfo_.accDividendsPerShare = accDividendsPerShare.add(toDistribute.mul(1e10).div(activeGrailSupply()));` However, if this `activeGrailSupply()` becomes a severely large value compared to `toDistribute` this will cause precision errors due to rounding.
Recommendation	Consider increasing the precision to 1e18 across the entire contract.
Resolution	RESOLVED Precision has been adjusted to 1e18 as recommended.

Issue #11	**`grailToken` can be made `immutable`**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the variable explicitly immutable.
Resolution	RESOLVED

Issue #12	Inconsistent usage of `reentrancyGuards`
Severity	INFORMATIONAL
Description	The contract contains a reentrancy guard on `harvestAllDividends` but such a guard is omitted on `harvestDividends` and `updateUser`. Even though we cannot find any reentrancy exploits, it shows inconsistency. If this was done intentionally it might be upon the consideration that `harvestAllDividends` is not strictly written in checks-effects-interactions and that they therefore added a `reentrancyGuard` as a general rule. However, the client should understand that only functions with such a guard are protected from reentrancy which means one could still reenter in any unprotected function, from `harvestAllDividends`. Again, Paladin did not find any reentrancy issues but we understand that the client might have wanted to rather be safe than sorry.
Recommendation	Taking the desire to be careful with reentrancy vectors in consideration, we recommend locking down `harvestDividends`, `updateUser` and optionally other functions with reentrancy guards.
Resolution	RESOLVED Reentrancy guards have been introduced on both functions.

Issue #13	Early return in `updateDividendsInfo` causes the "next cycle logic" to not always trigger
Severity	INFORMATIONAL

Description	The `updateDividendsInfo` contains the following early return:
	`if (activeGrailSupply() == 0 || currentBlockTimestamp < currentCycleStartTime) {`
	`  dividendsInfo_.lastUpdateTime = currentBlockTimestamp;`
	`  return;`
	`}`
	However, as `updateCurrentCycleStartTime();` is called before this, it can cause the cycle rollover code to not be triggered:
	`if (lastUpdateTime < currentCycleStartTime) {`
	This scenario is extremely unlikely as it would require `activeGrailSupply()` to equal to zero. But in this case, the logic seems to be wrongly defined.

Recommendation	Consider this scenario carefully. As Paladin understands the unlikelihood of there being an `activeGrailSupply` of exactly zero, it will be resolved based on this consideration or any addressing of it.
Resolution	RESOLVED The client understands this scenario and also sees this scenario as highly improbable.

Issue #14	Lack of SafeERC20 usage
Severity	INFORMATIONAL
Description	The contract presently does not use SafeERC20, which could cause issues if a fringe set of tokens were to be used as dividend tokens.
Recommendation	Consider using SafeERC20 and replacing the transfers with their safe equivalent, as it is already included in the contract.
Resolution	RESOLVED `safeTransfer` is now consistently used within this contract.

2.4 MasterChef

The MasterChef is a modified contract inspired by the Sushi MasterChef. On a functional level, it is extremely similar to the Sushi MasterChef but differs in the fact that it allows for two different reward tokens (GRAIL and EXC). It also introduces a deposit fee which can be set to a maximum of 4% for each individual pool and uses seconds instead of blocks to account for time. The deposit fee is sent to the fee address.

On a code-quality level, the MasterChef by Excalibur is written in a notably higher standard of quality compared to the original Sushi MasterChef. The Excalibur MasterChef's code is structured better, more secure against edge cases and protects the investor better. We commend Excalibur for taking the time to do this.

The contract initializes with a first single native pool. This first pool receives 800 `allocPoints` and allows users to deposit the Excalibur token to receive more Excalibur tokens.

2.4.1 Privileged Roles

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

- `setDevAddress`
- `setFeeAddress`
- `updateEmissionRate`
- `add`
- `set`
- `transferOwnership`
- `renounceOwnership`

2.4.2 Issues & Recommendations

Issue #15	Unused variable: `lpSupply`
Severity	INFORMATIONAL
Description	Variables defined in a contract but not used within said contract could confuse third-party auditors. They furthermore increase the contract length and bytecode size for no reason.
Recommendation	Consider renaming the lpSupplyWithMultiplier to lpSupply and only keeping track of lpSupply.
Resolution	RESOLVED The client indicates that they use `lpSupply` for UI purposes. It could also be used for extensions. They have thus kept this variable and Paladin agrees that this is reasonable.

Issue #16	`_excToken`, `_grailToken` and `startTime` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	RESOLVED

Issue #17	**Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `deposit`, `withdraw` and the pending rewards function, `accRewardsPerShare` is based upon the `pool.lpSupplyWithMultiplier` variable. `pool.accRewardsPerShare = pool.accRewardsPerShare.add(tokensReward.mul(1e12).div(pool.lpSupplyWithMultiplier));` However, if this `pool.lpSupplyWithMultiplier` becomes a severely large value this will cause precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have huge supplies and no decimals.
Recommendation	Consider increasing precision to 1e18 across the entire contract. It should be noted that even a precision of 1e18 has been considered small when tokens like PolyDoge were added to masterchefs of our client. In case the client thinks it's realistic that such tokens will be added we recommend testing which precision variable is most appropriate to support them without potentially reverting due to overflows.
Resolution	RESOLVED The precision has been increased to 1e18.

Issue #18	Inconsistent modifiers
Severity	INFORMATIONAL

Location	`Line 227`
	`function harvest(uint256 pid) external override nonReentrant validatePool(pid) {`
	
	`Line 241`
	`function deposit(uint256 pid, uint256 amount) external override validatePool(pid) nonReentrant`

Description	Order of modifiers is not just syntax sugar, it determines in which order they will be executed. Even though it does not affect any of the business logic, we recommend being consistent with modifier order to indirectly show to third-party reviewers that you understand that order matters.
Recommendation	Consider consistently ordering `nonReentrant` first and `validatePool` second.
Resolution	RESOLVED The recommendation has been introduced.

Issue #19	**Undetermined gas usage: At some point adding pools might run out of gas**
Severity	INFORMATIONAL
Description	Multiple functions within the MasterChef require the mass update function to be called. This means that under many pools, these functions might no longer work which means that governance can no longer add more pools. The result of this is that at some point no new pools can be added.
Recommendation	This issue will be resolved either by addressing it or by the remark that the client is fine with it.
Resolution	RESOLVED A `withUpdate` parameter was introduced to circumvent the gas issue if governance desires it.

2.5 MasterExcalibur

The MasterExcalibur extends the MasterChef with lock functionality. Specifically it extends upon the MasterChef base allowing users to lock their deposits for a period which can be set by the user. The user will receive a bonus equivalent to 50% of the harvested amounts if they opt to lock-in for the maximum lock period. However, if they, for example, only lock for half the maximum lock period, their bonus would only be 25% of the harvested amount. The bonus only unlocks at the end of the user-configured lock-in period unless the user extends their lock duration by either depositing into it again or extending it explicitly. Finally, the user has the option to deposit from an existing normal deposit to avoid having to pay the deposit fee twice; similarly, the user can withdraw to a normal deposit to avoid having to pay the deposit fee twice.

This contract inherits all functionality, issues and privileged operations from the MasterChef contract. The issues from the MasterChef contract will not be repeated within this section to keep the report readable.

2.5.1 Privileged Roles

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

- `transferOperator`
- `setDisableLockSlot`

2.5.1 Issues & Recommendations

Issue #20	**Users can receive a 50% bonus while being able to withdraw at any time**
Severity	INFORMATIONAL
Description	After the lock is fully vested, users can continue to enjoy the rewards as long as they do not harvest it. They can for example wait another week with harvesting after they need to unlock and this whole week would still be at a +50% bonus.
Recommendation	Consider whether this behavior would be problematic. If not this issue will be resolved on that note, if it is, it could be considered to cap the last harvest to the unlock date.
Resolution	RESOLVED The client has indicated this is desired behavior as they would like to incentivize users to keep their tokens staked longer.

Issue #21	**Inconsistent usage of `_msgSender()`**
Severity	INFORMATIONAL
Description	Within the `onlyOperator` function, `_msgSender()` is used. However, this pattern is used in no other locations of the contract. Using the overloadable `_msgSender()` function only makes sense if you use it consistently.
Recommendation	Consider being consistent with using `_msgSender` or not using it at all.
Resolution	RESOLVED

Issue #22	**`pendingRewardsOnLockSlot` lacks a `validateSlot` modifier and can revert if users have a zero deposit amount**
Severity	INFORMATIONAL
Description	The `pendingRewardsOnLockSlot` lacks a `validateSlot` modifier and reverts due to a division by zero if `userSlot.amountWithMultiplier` is zero. This case is extremely unlikely but could occur if the deposit or transfer tax fee causes the deposit to go from "1" to "0".
Recommendation	Consider adding a `validateSlot` modifier and returning early if the `amountWithMultiplier` is zero.
Resolution	RESOLVED

2.6 FeeManager

The FeeManager is a utility contract which can receive ERC-20 fees from the AMM and MasterChef. It can convert these tokens to the dividend tokens and distribute the new tokens as partially as dividends and partially to other locations.

The contract definition indicates that the funds will be sent to:

- Dividends contract: MIN 50%
- dev address: MAX 20%
- SAFU funds address: MIN 2%, MAX 10%
- buy back & burn address: MIN 5% MAX 25%

Initially the distribution is specified as follows:

- Dividends contract: 70%
- dev address: 20%
- SAFU funds address: 5%
- buy back & burn address: 5%

It should be noted that even though these limits are enforced in the contract, a malicious owner could just swap the contract balances to a token where he is the only owner of the LP liquidity of, and then remove his liquidity afterwards to capture 100% of the fees. As this would harm their reputation exceptionally and as Paladin is under the impression that this team would not seek such short term profit, this has not been included as an explicit issue. It is, however, a possibility and as private keys can be stolen, there is a non-zero chance of it happening.

Anyone can force distribution of registered tokens by calling `distributeFees` or `distributeFeesByToken`. Any Excalibur in the contract will be burned at this point. It should be noted that the owner should be careful not to use a dividend token as an

intermediary swap step as an unprivileged user might distribute these as soon as they are registered in the FeeManager.

2.6.1 Privileged Roles

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

- `updateShares`
- `initializeRouter [ callable once ]`
- `removeLiquidityToToken`
- `removeAllLiquidityToToken`
- `swapBalanceToToken`
- `setDevAddr`
- `transferOwnership`
- `renounceOwnership`

2.6.2 Issues & Recommendations

Issue #23	**updateShares wrongly checks the validity of buybackAndBurnShare_ causing it to be uncallable by governance**
Severity	HIGH SEVERITY

Location	Lines 114-121
	`require(`
	`  buybackAndBurnShare_ <= MIN_BUYBACK_AND_BURN_SHARE,`
	`  "FeeManager: buybackAndBurnShare mustn't exceed minimum"`
	`);`
	`require(`
	`  buybackAndBurnShare_ >= MAX_BUYBACK_AND_BURN_SHARE,`
	`  "FeeManager: buybackAndBurnShare mustn't exceed maximum"`
	`);`

Description	The buybackAndBurn validation logic has been accidentally inverted in updateShares — this causes this function to always revert.
Recommendation	Consider inverting the inequalities.
Resolution	RESOLVED

Issue #24	**Lack of minimumReceived parameter in swaps might allow for frontrunning**
Severity	INFORMATIONAL
Description	swapBalanceToToken currently allows for infinite slippage — this could allow for large swaps to be sandwiched.
Recommendation	Consider adding a minAmountReceived parameter to swapBalanceToToken.
Resolution	RESOLVED

Issue #25	**Gas optimization: Unnecessary addition within uniswap router operations**
Severity	INFORMATIONAL
Description	The Uniswap router operations presently add 100 seconds to the timestamp — this is not necessary as `block.timestamp` can be used. In addition, using `type(uint256).max` might result in gas savings as it can be hardcoded in the bytecode and would therefore be nearly free of charge in gas cost terms. Finally, it might make sense to set the minimum amounts received to 1 as there is no point in removing liquidity/swapping/... if it doesn't at least give you 1 token.
Recommendation	Consider setting the deadline to the maximum uint256 and adding 1 as the minimum for all Uniswap operations.
Resolution	RESOLVED

Issue #26	**`excToken`, `dividendsContract`, `safundsAddress` and `buybackAndBurnAddress` can be made `immutable`**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	RESOLVED

2.7 ERC20BurnSupply

The ERC20BurnSupply contract extends the standard OpenZeppelin ERC-20 implementation with a `burnSupply()` function which shows the number of tokens burned.

2.7.1 Issues & Recommendations

No issues found.

2.8 ERC20AvgReceiveTime

The ERC20AvgReceiveBTimecontract extends the standard OpenZeppeling ERC-20 implementation with accounting functionality that keeps track of the average token receipt time. If 100 tokens were received at timestamp 5, and 20 at timestamp 50, the average time would be timestamp 6. This average timestamp, which is kept for every account that holds tokens, can be used in derivative contracts for extra functionality. Specifically, it is used within the WrapERC20WithPenalty contract.

2.8.1 Issues & Recommendations

No issues found.

2.9 WrapERC20WithPenalty

TheWrapERC20WithPenalty contract extends the ERC20AvgReceiveTimestamp contract with conversion functionality that allows the extended token to be "unwrapped" into newly minted tokens of a different kind (the "regular" token). However, this conversion happens at a rate which can be less than 1:1. The rate worsens as the average token receipt time is more recent. The rate improves linearly over the configured penalty period, starting at a configured maximum conversion penalty going all the way to a minimum conversion penalty.

The documentation seems to indicate that these parameters will be set at 70% to 90% and that this contract will be exclusively used for GRAIL to convert GRAIL into newly minted Excalibur.

2.9.1 Issues & Recommendations

Issue #27	Lack of constructor validation
Severity	INFORMATIONAL
Description	Presently, the constructor does not validate that the `_unwrapPenaltyMin` is smaller or equal to the `_unwrapPenaltyMax`. It also does not validate that both these parameters are smaller or equal to 100.
Recommendation	Consider adding requirements to enforce the above properties.
Resolution	RESOLVED

2.10 Multicall

The Multicall is a simple function batcher based on the Maker Multicall contracts. It is solely used for frontend purposes.

2.10.1 Issues & Recommendations

No issues found.

2.11 ExcaliburV2Factory

The ExcaliburV2Factory is a fork of Uniswap's UniswapV2Factory contract. It is in charge of managing all the existing asset pairs and allows users to create new pairs if matched tokens for supplying liquidity have no existing contract pair. The ExcaliburV2Factory deploys UniswapV2Pair contracts.

2.11.1 Privileged Roles

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

- `setOwner`
- `setFeeTo`
- `setOwnerFeeShare [ up to 50% ]`
- `setReferrerFeeShare [ up to 20% ]`

2.11.2 Issues & Recommendations

Issue #28	Lack of events for `setOwner` and `setFeeToSetter`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.12 UniswapV2Pair

The UniswapV2Pair is involved in storing the asset pairs in a contract for use by the router to add and remove liquidity in equally-valued proportions and for swapping assets. It is a fork of the Uniswap version of this contract but extends it with the following features.

First, the swap fee is configurable. It is initially set to 0.15% for all pairs but can be configured on a per-pair basis to be between 0.01% to 2% by the governance. This is done through the `setFeeAmount` function. It also allows for a dynamic percentage of the fees to go to the governance. This percentage is, however, configured globally.

Next, it allows for each user that swaps to provide a referrer address during their swap — this referer receives a configurable percentage of the swap fee, which is configured globally per referrer by the governance. This could allow Excalibur to strategically partner with vaults and other projects to use their AMM and get a kickback. The fee can be up to 20% of the swap fee.

Third, it improves upon the traditional Uniswap Pair by only letting `sync()` be called if there is a valid ratio. This prevents an annoying exploit some owners have experienced where a malicious party sends one of the tokens to the pair before liquidity is added and calls `sync()`, thereby preventing the owner to add liquidity through the normal user interface (which in fact crashes in this scenario due to a division by zero).

Finally, governance can take any token out of the pair except for the main tokens. Since the main tokens cannot be touched by governance, this is considered innocent.

2.12.1 Privileged Roles

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

- `setFeeAmount`
- `drainWrongToken`

2.12.2 Issues & Recommendations

Issue #29	**Setting the ownerFeeShare to zero prevents all minting and breaking of LP tokens**
Severity	HIGH SEVERITY
Location	Line 122 `uint d = (FEE_DENOMINATOR / IExcaliburV2Factory(factory).ownerFeeShare()).sub(1);`
Description	The LP pairs contain logic for a dynamic percentage of the fee to go to the governance wallet. This percentage can be configured in the factory by the governance. However, if this fee were to be set to zero, this causes all mints and burns to break and therefore nobody would be able to break up their LP pairs.
Recommendation	Consider setting `d` to zero if the `ownerFeeShare()` is zero. We recommend caching `FEE_DENOMINATOR / IExcaliburV2Factory(factory).ownerFeeShare()` in a temporary variable and check if this is non-zero to save on gas.
Resolution	RESOLVED `ownerFeeShare` must now be non-zero — this is enforced in the setter.

Issue #30	Protocol fee will be slightly less than expected
Severity	INFORMATIONAL
Description	The part of the swap fee which is given to the protocol will be slightly less than expected due to the referrer fee. We explain this with an example: Assume a referrer fee of 20% and protocol fee of 50%: 1. Swap occurs — 20% of the fee goes to the referrer; 80% of the fee goes to increasing K. 2. 50% of the increase in K is minted to the protocol, eg. 40% of the swap fee. In this example, the protocol configured a swap fee of 50% but in fact only received 40%. Although this is really innocent, we think it is good that the protocol understands this.
Recommendation	Consider taking this in consideration when setting the fee levels.
Resolution	RESOLVED The client has indicated that they will take this in consideration when setting the fee levels.

Issue #31	Lack of events for `drainWrongToken`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	RESOLVED

Issue #32	Gas optimization: Lack of `calldata` usage on `internal` function
Severity	INFORMATIONAL
Description	If an internal function is solely called by external functions, it can have `calldata` parameters as well.
Recommendation	Consider replacing `memory` with `calldata` on `_swap`.
Resolution	RESOLVED The client has indicated their Solidity version does not allow for this behavior yet.

2.13 UniswapV2Erc20

The UniswapV2Erc20 contract is an implementation of the ERC-20 Token Standard for denominating pool tokens. It is a fork of Uniswap's UniswapV2ERC20 contract.

2.13.1 Issues & Recommendations

Issue #33	**Approval event is not emitted if allowance is changed in `transferFrom` as suggested in the ERC-20 Token Standard (also present in Uniswap)**
Severity	INFORMATIONAL
Description	The ERC-20 standard specifies that an approval event should be emitted when the allowance of a user changes. However, within the ERC20 implementation of both Uniswap and Excalibur, this is not done. You can read more about this improvement in Pull Request #65 of uniswap-core.
Recommendation	Consider adding `emit Approval(from, msg.sender, remaining)` in `transferFrom` when allowance is modified.
Resolution	RESOLVED An approval event is now emitted according to the PR.

Issue #34	**`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)**
Severity	INFORMATIONAL
Description	Currently if `permit` is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up permit transactions in the mempool and execute them before a contract can. The implications of this issue is that a bad actor could prevent a user from removing liquidity with a permit through the router. It is a denial of service attack which is present in all AMMs but which we have yet to witness being used since there is no profit from it.
Recommendation	Consider this issue if there are ever complaints by users that their `removeLiquidityWithPermit` transactions are failing. It could be the case that someone is using this vector against them. We do not recommend changing this behavior since it would cause a lot of extra work modifying the frontend to account for the new `permit` behavior. This issue is also present in Uniswap after all.
Resolution	RESOLVED The client has indicated that they will take this into consideration.

2.14 Math, SafeMath, UQ112x112

Math, SafeMath and UQ112x112 are various helper libraries which are each identical to the Uniswap implementation.

2.14.1 Issues & Recommendations

No issues found.

2.15 ExcaliburRouter

2.15.1 Privileged Roles

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

- `setFeeRebateDisabled`
- `setMaxDailyEXCAllocation`
- `transferOperator`

2.15.2 Issues & Recommendations

Issue #35	**Fee rebate mechanism uses pair prices which can be easily manipulated and lacks robustness**
Severity	HIGH SEVERITY
Description	Presently the fee rebate mechanism mints EXC to the swapper based on the amount of fees they paid for their swap. This mechanism is quite brittle as one might be able to swap for basically free if they own the whole pair and then dump the rebate perpetually, until EXC is valued so low that it is no longer worth it. If tokens are mispriced, an exploiter could potentially abuse this as well. Finally, because pair prices are used, an exploiter can likely abuse these too as they are easily manipulated in sandwich attacks.

Recommendation	Ideally the system should address two concerns:
	
	1. Pair prices should not be used as they can be manipulated easily
	
	2. As exploits are difficult to perfectly avoid in a setup like this, the system must be robust if someone can find a profitable arbitrage opportunity. The result of this should not be a huge dump of the native token.
	
	Our recommendation is to move away from perfectly rebating fees and go for a mechanism where a fixed number of Excalibur is distributed each day for rebates. The following pseudo-code, which lacks a huge deal of safeguards (eg. the fact that claiming can happen over and over again) should serve as inspiration.
	
	`uint256 public dailyEXC = 50 000 ether;`
	`mapping(uint256 => (address => uint256)) public usdMinedInDay;`
	`mapping(uint256 => uint256) public totalUsdMinedInDay;`
	`function receiveForDay(uint256 day) external {`
	`_mint(msg.sender, dailyEXC * usdMinedInDay[day]`
	`[msg.sender] / totalUsdMinedInDay[day]);`
	`}`

Resolution RESOLVED

Although the system is still theoretically vulnerable to such exploits, the client has made significant resolutions to reduce the likelihood and impact of them. The most ingenious design decision is that the EXC price used is non-decreasing, which means that although it still uses the pair price, if an exploiter were to manipulate the price of that pair to be lower, it would not affect the mechanism price. Secondly, the client has indicated that they will solely use pairs which have a ChainLink quote and will also also quote them at the minimum value of the ChainLink quote and their on-chain quote. This values any swap at a lower rebate value than it originally would be valued, taking the most pessimistic view. This should significantly reduce the chances of exploits.

However, as anything can still be whitelisted for rebates, exploits cannot be excluded completely. It only requires an exploiter to find a single moment where an arbitrage would be possible to potentially exploit the rebate mechanism and mint and dump tokens. Through discussions with Paladin, the client also understands that this is something worth addressing and has therefore included daily mint limits within the system which means that every day a limit of tokens that can be granted as rebates is instated. It is up to the client to keep this limit at a reasonable level so that if an exploit were to ever occur, this does not severely impact the project.

Issue #36	Typographical error: Contract defines the USD price as BUSD in the variable names
Severity	LOW SEVERITY
Description	The contract defines the USD price as the BUSD price in the variable names. However, this is actually the ChainLink USD quote of the asset, which is not necessarily denominated in BUSD.
Recommendation	Consider simply using USD.
Resolution	RESOLVED The code section has been removed.

Issue #37	Governance privilege: price consumer can be used or changed to potentially mint excessive excalibur tokens
Severity	INFORMATIONAL
Description	Governance has the ability to adjust the price consumer which is essentially the oracle to derive the fee rebates through. This can be used to mint excessive amounts of the native token by making the native coin worth $0 or some dummy token only governance owns worth millions. This would cause them to receive excessive fee rebates to potentially mint and dump the token.
Recommendation	Consider eventually timelocking all functionality except the pausing of the rebate mechanism. Also the functionality within the oracle itself should be timelocked eventually. We understand the need to have flexibility over these critical sections of code, especially in the early days, so do understand if this issue is initially acknowledged.
Resolution	ACKNOWLEDGED The client has separated the role for this functionality to allow it to eventually be timelocked. We agree with the client that having some flexibility over this functionality early on is desired.

Issue #38	**Adding logic to the fallback function reduces the limited gas stipend of WETH withdrawals which could make them more likely to revert under protocol upgrades**
Severity	INFORMATIONAL
Description	There's no explicit commitment that gas costs will always be the same. However, WETH withdrawals already consume most of the allocated gas as WETH uses .send which has a very limited stipend. By adding more logic in the fallback function `receive()`, you reduce this wiggle room further and increase the likelihood that under gas adjustments, your protocol would stop working. This issue is marked as informational since this code is present in Uniswap as well. We doubt any reasonable protocol developer would want to break Uniswap.
Recommendation	Consider removing the code within the `receive()` function.
Resolution	RESOLVED

2.16 PriceConsumerV3

The PriceConsumerV3 is a price oracle which uses safe ChainLink functionality for part of its functions and unsafe pair calls for others. `getTokenFairPriceUSD` and `getWETHFairPriceUSD` use ChainLink and are therefore safe from obvious manipulation.

2.16.1 Privileged Roles

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

- `setOwner`
- `setWhitelistToken`
- `setTokenPriceFeeder`

2.16.2 Issues & Recommendations

Issue #39	**`_getWETHFairPriceUSD` and `_getTokenFairPriceUSD` do not revert if the price is negative or stale**
Severity	LOW SEVERITY
Description	Presently the get price functions do not revert if a problem occurs with chainlink and prices turn negative. There is also no logic that sets the price to zero if it is stale.
Recommendation	Consider reverting if the price is negative and setting the price to zero if stale. Alternatively, the price can also be set to zero if negative.
Resolution	RESOLVED In these scenarios, a price of 0 is returned, which is acceptable within the audit scope since these assets should always remain "underpriced" by the oracle to prevent exploits.

Issue #40	**Typographical error: Contract defines the USD price as BUSD in the variable names**
Severity	LOW SEVERITY
Description	The contract defines the USD price as the BUSD price in the variable names. However, this is actually the ChainLink USD quote of the asset, which is not necessarily denominated in BUSD.
Recommendation	Consider simply using USD.
Resolution	RESOLVED

Issue #41	`_getTokenPriceUSDUsingPair` does not properly handle decimals
Severity	LOW SEVERITY
Description	Presently the `_getTokenPriceUSDUsingPair` function only handles the input token decimals and not the output (quote) token decimals.

Recommendation	Consider addressing this problem in steps:
	1. We want to quote 1 input token.
	`uint256 inputAmount = 10 ** IERC20(token).decimals();`
	2. We want to convert it to output (quote) tokens (note that this conversion is prone to manipulation and does not account for slippage).
	`uint256 quoteAmount = inputAmount * reserveOut / reserveIn;`
	3. We want to convert the output amount to 18 decimals.
	`uint256 priceInQuote = quoteAmount * 1e18 / (10 ** IERC20(token.decimals());`
	This last step can be optimized for gas with an if-else statement.
	It should be noted that we feel like the quote amount should simply be left in the decimals of the quote token.

Resolution	RESOLVED The logic has been updated to handle decimals properly.

Issue #42	**`getTokenFairPriceUSD` and `getWETHFairPriceUSD` can be made external**
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	RESOLVED

2.17 UniswapV2Library

The UniswapV2Library is a contract used to calculate some common calculations like the amount to receive from a swap. Excalibur has slightly modified it to work with the variable swap fee.

2.17.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY