PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For PolyShield Finance

14 August 2021


paladinsec.co


info@paladinsec.co

Table of Contents

Disclaimer

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

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

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

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

1 Overview

This report has been prepared for PolyShield Finance. 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	PolyShield Finance
URL	https://polyshield.finance
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`MasterChef`	`0x0Ec74989E6f0014D269132267cd7c5B901303306`	MATCH
`Polyshield (Token)`	`0xf239e69ce434c7fb408b05a0da416b14917d934e`	MATCH
`PolyshieldBurner`	`0xFbb307Fea8CdaF614B66f82D8d233c947B07c4F5`	MATCH
`Timelock`	`0xa63C2e89441023DEC0258736531Da27af6230B19`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	1	-	1
Medium	0	-	-	-
Low	10	1	-	9
Informational	14	-	-	14
Total	**26**	**2**	**0**	**24**

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 MasterChef

ID	Severity	Summary	Status
01	HIGH	Flash loan allows an exploiter to set the emission rate through LP pair price manipulation	RESOLVED
02	HIGH	Severely excessive rewards issue when a token with a transfer tax is added (native tokens are handled correctly)	ACKNOWLEDGED
03	LOW	Setting `feeAddress` to the zero address will break most functionality	ACKNOWLEDGED
04	LOW	Lack of safeguards on `topPrice`, `bottomPrice` and `curveRate` could lead to reverting transactions or excessive rewards if these are set incorrectly by the governance	ACKNOWLEDGED
05	LOW	The emission rate curve behavior might be undesirable if the bottom rate is set to a higher value	ACKNOWLEDGED
06	LOW	Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`	ACKNOWLEDGED
07	LOW	The `pendingShield` function will revert if `totalAllocPoint` is zero	ACKNOWLEDGED
08	LOW	`updateEmissionRate` has no maximum safeguard	ACKNOWLEDGED
09	LOW	`updateStartBlock` does not update `PoolInfo.lastRewardBlock` and is callable after farm launch	ACKNOWLEDGED
10	INFORMATIONAL	Loss of precision due to division before multiplication in the `updateEmissionIfNeeded` function	ACKNOWLEDGED
11	INFORMATIONAL	Rounding vulnerability of tokens with a very large supply can cause them to receive zero emissions	ACKNOWLEDGED
12	INFORMATIONAL	Gas optimization: `harvestAll` transfer only has to be done once	ACKNOWLEDGED
13	INFORMATIONAL	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
14	INFORMATIONAL	`BONUS_MULTIPLIER` is not actively used	ACKNOWLEDGED
15	INFORMATIONAL	`add`, `set`, `deposit`, `harvest`, `harvestAll`, `withdraw`, `emergencyWithdraw`, `setFeeAddress`, `getShieldPriceCents`,`getEmissionRatePercent`, `updateEmissionParameters`, `updateEmissionRate`, `updateStartBlock`, `setNFTAddress`, `setUSDCAddress`, and `setUsdcShieldLPAddress` functions can be made external	ACKNOWLEDGED
16	INFORMATIONAL	`msg.sender` is already an address and does not have to be cast to address again	ACKNOWLEDGED
17	INFORMATIONAL	`polyShield` can be made immutable	ACKNOWLEDGED
18	INFORMATIONAL	`maxEmissionRate` and `burnaddr` can be made constant	ACKNOWLEDGED

19	INFORMATIONAL	`curveRate` can overflow in `int256` cast	ACKNOWLEDGED
20	INFORMATIONAL	Lack of events for `add`, `set`, `updateEmissionIfNeeded`, `updateEmissionParameters`, `updateStartBlock`, `setNFTAddress`, `setUSDCAddress` and `setUsdcShieldLPAddress`	ACKNOWLEDGED

1.3.2 PolyShield

ID	Severity	Summary	Status
21	LOW	`mint` function could have been used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	RESOLVED
22	LOW	`minted` and `burned` variables can overflow causing the variables to misrepresent the actual minted and burned values if extremely large amounts are minted or burned	ACKNOWLEDGED
23	INFORMATIONAL	`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `updateEmissionRate` and `setFeeAddress` can be made external	ACKNOWLEDGED

1.3.3 TokenBurner

ID	Severity	Summary	Status
24	LOW	`burnTotals` can overflow, causing the variables to misrepresent the actual minted and burned values if extremely large amounts are minted or burned	ACKNOWLEDGED
25	INFORMATIONAL	`polyshield` can be made `immutable`	ACKNOWLEDGED
26	INFORMATIONAL	`burn` can be made `external`	ACKNOWLEDGED

1.3.4 Timelock

No issues found.

2 Findings

2.1 MasterChef

The Masterchef is a fork of Goose Finance's Masterchef. A notable feature of forking this Masterchef is the removal of the `migrator` function from PancakeSwap, which of late has been used maliciously to steal users' tokens. Furthermore, in comparison to Goose Finance, PolyShield has limited the deposit fee to at most 4%. We commend Polyshield on their decision to fork a relatively safer version of the Masterchef.

The Masterchef includes some interesting and notable modifications. First, the emission rate is automatically adjusted based on the token price. Second, an `nftaddress` could be defined at a later point in time which would then receive about 5% of the minted rewards, presumably as some special feature. Finally, the Masterchef contains utility functions to harvest all pools within a single call.

2.1.1 Privileged Roles

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

- `renounceOwnership`
- `transferOwnership`
- `add`
- `set`
- `setFeeAddress`
- `updateEmissionParameters`
- `updateEmissionRate`
- `updateStartBlock`
- `setNFTAddress`
- `setUSDCAddress`
- `setUsdcShieldLPAddress`

2.1.2 Issues & Recommendations

Issue #01	**Flash loan allows an exploiter to set the emission rate through LP pair price manipulation**
Severity	HIGH SEVERITY
Description	Currently the emission rate is adjusted every 30 minutes based on the current instantaneous price in the LP pair of the PolyShield token. To do this, a simple division between the two balances in the pair is made to get the price. Using a flash loan, an exploiter is able to temporarily withdraw tokens from the pair as long as they are returned in the same transaction with an approximately 0.3% fee. As of now, about $160k USDC is in the relevant pair, so it would cost approximately $480 to take this all out (and thus set the price to near 0). The exploiter could pay the same cost to temporarily take out all PolyShield tokens and set the price to near infinity. Thus, a malicious party can, at a cost of about $500 per half hour (the current emission update interval), set the emission rate from 0 all the way to 1 token per block.
Recommendation	The solution that solves this issue completely is using a TWAP (time-weighted average price) which is just the average price over a previous set of intervals. Uniswap is able to generate such prices easily. However, since the contracts are already deployed, implementing this solution would likely require a redeploy. Thus, other solutions that could be considered are: 1. Simply acknowledge the issue. 2. Reduce the `emissionUpdateInterval` to a very low value which makes this issue on average much less profitable to an exploiter. 3. Disable the automatic emission rate mechanism by setting `usdcShieldLP` to the zero address.
Resolution	RESOLVED They have reduced the `emissionUpdateInterval` to only 5 minutes now, thus reducing the incentive for any exploits.

Issue #02	**Severely excessive rewards issue when a token with a transfer tax is added (native tokens are handled correctly)**
Severity	HIGH SEVERITY
Description	When tokens with a transfer tax are added to the pools, this will result in significant excessive rewards. Due to the way the Masterchef handles rewards, rewards can be heavily inflated when the balance of the Masterchef no longer matches that of user deposits. This happens for example with transfer tax tokens. This issue is further amplified on Masterchefs like this one with a referral mechanism, since tokens can be minted directly. This flaw of the Masterchef has recently been exploited on a significant number of projects, all of which their native tokens went to $0 afterwards because the exploit resulted in a large number of native tokens being minted and dumped. This issue was also present in SushiSwap (the original Masterchef). Since they were never meant to have any tokens but LP tokens, it was not a problem there but has become a problem to projects who have started forking it for usage with less standard tokens. Note that the native token is handled correctly since there is explicit logic. However, this logic is not disabled for depositors which are excluded from the transfer tax. This will however not cause any severe side effects since the Masterchef simply receives 2% tokens extra from these users.
Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:

```
uint256 balanceBefore = pool.lpToken.balanceOf(address(this));
pool.lpToken.transferFrom(msg.sender, address(this), _amount);
_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);
```

Resolution	ACKNOWLEDGED The team has stated they will not add tokens with transfer taxes.

Issue #03	**Setting `feeAddress` to the zero address will break most functionality**
Severity	LOW SEVERITY
Description	Transferring tokens to the zero address will revert transactions. Most deposits and withdrawals will thus revert if the `feeAddress` is ever set to the zero address. `harvest` and `harvestAll` calls would break as well in this instance.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like `require(_feeAddress != address(0), "!nonzero");` to the `setFeeAddress` function.
Resolution	ACKNOWLEDGED The team has stated they will implement a safeguard in their upcoming governance contract. We shall mark this as resolved when we confirm that has occured.

Issue #04	**Lack of safeguards on `topPrice`, `bottomPrice` and `curveRate` could lead to reverting transactions or excessive rewards if these are set incorrectly by the governance**
Severity	LOW SEVERITY
Description	The `updateEmissionParameters` function which is responsible for setting the top price, bottom price and curve rate of the price-emissions curve lacks safeguards. This could for example cause the emissions calculations to revert if `bottomPrice` is set to a value lower than 1% of the top price (division by zero in FixidityLib.divide call) or the curve to not span in the percentage range [0%, 100%] but potentially exceed it.
Recommendation	Consider always simulating all possible `shieldPriceCent` variables before updating the `updateEmissionParameters`. In this case, that simulation would require 1000 data-points which is very feasible to generate using web3. Note that the `getEmissionRatePercent` function is extremely fragile (overflows, divisions by zero, undesirable behavior) in general and thus we recommend to consider the comments above seriously and always go through all values of the curve before updating it.
Resolution	ACKNOWLEDGED The team has stated that this is a desired feature, and that they will implement a safeguard in their upcoming governance contract. We shall mark this as resolved when we confirm that has occured.

Issue #05	**The emission rate curve behavior might be undesirable if the bottom rate is set to a higher value**
Severity	LOW SEVERITY
Description	The bottom rate variable is simply used as a cutoff value and does not actually move the curve up when it is set to a higher value. If the maximum price were for example $1000 and the minimum price $10, this could create a large drop when the price goes from $10.1 to $9.9.
Recommendation	Consider whether this sudden drop at larger values is desirable and if not, consider moving the curve explicitly by working on the `[bottomPrice, topPrice]` range instead of the `[0, topPrice]` range.
Resolution	ACKNOWLEDGED The team has stated they will implement a safeguard in their upcoming governance contract. We shall mark this as resolved when we confirm that has occured.

Issue #06	**Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate`**
Severity	LOW SEVERITY
Description	`updatePool` will always call `balanceOf(address(this))` on the token of this pool, and will fail if the token is not an actual token contract address.
Recommendation(s)	Consider simply adding a test line in the add function. If the token does not exist, this will make sure the add function fails. `_lpToken.balanceOf(address(this));`
Resolution	ACKNOWLEDGED The team has stated they will implement a safeguard in their upcoming governance contract. We shall mark this as resolved when we confirm that has occured.

Issue #07	**The `pendingShield` function will revert if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	In the `pendingShield` function, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error.
Recommendation	Consider only calculating the accumulated rewards since the `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so: `if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	ACKNOWLEDGED The team has stated that they will not set allocPoints to zero.

Issue #08	**updateEmissionRate has no maximum safeguard**
Severity	LOW SEVERITY
Description	Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent.
Recommendation	Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value. `require(_shieldPerBlock <= MAX_EMISSION_RATE,"Too high");`
Resolution	ACKNOWLEDGED The team has stated they will implement a safeguard in their upcoming governance contract. We shall mark this as resolved when we confirm that has occured.

Issue #09	**updateStartBlock does not update PoolInfo.lastRewardBlock and is callable after farm launch**
Severity	LOW SEVERITY
Description	When users deposit in a pool, they will start earning rewards as soon as the current block is larger than the pools' `lastRewardBlock`. When a pool is added, this `lastRewardBlock` is set to either the `currentBlock` or the `startBlock` (whichever is greater). If the `startBlock` is moved, the `lastRewardBlock` absolutely have to be moved as well. There have been multiple cases of projects that have failed to start as their `lastRewardBlock` was still weeks in the future, even though they called `updateStartBlock`. This issue can cause significant reputational damage. Since the farm has already started we decided to move this issue to low risk.
Recommendation	Consider adding a loop to update all lastRewardBlocks in the updateStartBlock function. Note that this loop may run out of gas if a very significant amount of pools are added.

```
function updateStartBlock(uint256 _startBlock) external onlyOwner {
  require(startBlock > block.number, "Farm already started");
  require(_startBlock > block.number, "Farm already started");
  uint256 length = poolInfo.length;
  for (uint256 pid = 0; pid < length; ++pid) {
    PoolInfo storage pool = poolInfo[pid];
    pool.lastRewardBlock = _startBlock;
  }
  startBlock = _startBlock;
}
```

Resolution	ACKNOWLEDGED The team has stated they will implement a safeguard in their upcoming governance contract. We shall mark this as resolved when we confirm that has occured.

Issue #10	**Loss of precision due to division before multiplication in the `updateEmissionIfNeeded` function**
Severity	INFORMATIONAL
Location	Line 312 `shieldPerBlock = maxEmissionRate.div(100).mul(emissionRatePercent);`
Description	Within the `updateEmissionIfNeeded` function, division is done before multiplication which results in a lower precision due to Solidity rounding down. It should be noted that this loss of precision is insignificant due to `maxEmissionRate` being a large number and the division only dividing by 100.
Recommendation	Consider doing multiplication before division.
Resolution	ACKNOWLEDGED

Issue #11	**Rounding vulnerability of tokens with a very large supply can cause them to receive zero emissions**
Severity	INFORMATIONAL
Description	Within `updatePool`, `accShieldPerShare` is based upon the `lpSupply` variable. `accShieldPerShare = accShieldPerShare.add(shieldReward.mul(1e12).div(lpSupply));` However, if this `lpSupply` becomes a severely large value, there will be precision errors due to rounding. This is famously seen when pools decide to add meme-tokens which usually have a huge supply and no decimals.
Recommendation	Consider increasing precision to 1e18 across the entire contract.
Resolution	ACKNOWLEDGED

Issue #12	Gas optimization: `harvestAll` transfer only has to be done once
Severity	INFORMATIONAL
Description	During the `harvestAll` function, a transfer call is made for every single pool. It would be more gas efficient however to simply accumulate the pending amount in a variable and transferring it out once at the end of the `harvestAll` method.
Recommendation	Consider accumulating the pending rewards and transferring it once by: 1. Moving `uint256 pending = 0;` outside of the for loop 2. Changing the pending sets to pending increments with SafeMath. 3. Moving the `safeShieldTransfer (with pending > 0 check)` to the end.
Resolution	ACKNOWLEDGED

Issue #13	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs, the total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool.
Recommendation	Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. Each `lpToken.balanceOf(address(this))` query can then be replaced with this `lpSupply` as well.
Resolution	ACKNOWLEDGED

Issue #14	**`BONUS_MULTIPLIER` is not actively used**
Severity	INFORMATIONAL
Description	The constant variable `BONUS_MULTIPLIER` does not contain any extra information since it is constant and cannot be changed from 1. The public `shieldPerBlock` variable does indicate all information that is necessary to understand the current emission rate.
Recommendation	Consider removing `BONUS_MULTIPLIER`.
Resolution	ACKNOWLEDGED

Issue #15	**`add`, `set`, `deposit`, `harvest`, `harvestAll`, `withdraw`, `emergencyWithdraw`, `setFeeAddress`, `getShieldPriceCents`, `getEmissionRatePercent`, `updateEmissionParameters`, `updateEmissionRate`, `updateStartBlock`, `setNFTAddress`, `setUSDCAddress`, and `setUsdcShieldLPAddress` functions can be made `external`**
Severity	INFORMATIONAL
Description	The above functions can be changed from `public` to `external`. 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 making these functions external.
Resolution	ACKNOWLEDGED

Issue #16	`msg.sender` is already an address and does not have to be cast to address again
Severity	INFORMATIONAL
Description	Throughout the codebase, `msg.sender` is explicitly cast to an address like in the following code: `address(msg.sender)`. Since `msg.sender` is already an address, this code is redundant (although it does no harm either).
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	ACKNOWLEDGED

Issue #17	`polyShield` 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 `polyShield` explicitly `immutable`.
Resolution	ACKNOWLEDGED

Issue #18	`maxEmissionRate` and `burnaddr` can be made constant
Severity	INFORMATIONAL
Description	Constant variables can be explicitly declared as `constant`. This is best practice as it simplifies the reviewing process for third-party reviewers and saves gas. Static-analysis tools could also take this into consideration.
Recommendation	Consider making these variables constant.
Resolution	ACKNOWLEDGED

Issue #19	`curveRate` can overflow in `int256` cast
Severity	INFORMATIONAL
Description	The `curveRate` can overflow when it is cast to an `int256`. This is because the `curveRate` is saved as an `uint256` but is later cast to an `int256`. This allows for the governance to inject a negative `curveCoefficient`.
Recommendation	Consider making these variables constant.
Resolution	ACKNOWLEDGED

Issue #20	**Lack of events for `add`, `set`, `updateEmissionIfNeeded`, `updateEmissionParameters`, `updateStartBlock`, `setNFTAddress`, `setUSDCAddress` and `setUsdcShieldLPAddress`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

2.2 PolyShield

The libraries have not yet been verified.

2.2.1 Token Overview

Address	0xF239E69ce434c7Fb408b05a0Da416b14917d934e
Token Supply	Initial supply: 1000 tokens Max supply: ∞
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	None

2.2.2 Privileged Roles

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

- `mint`

2.2.3 Issues & Recommendations

Issue #21	**`mint` function could have been used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef**
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint tokens for dumping. As token ownership has already been transferred to the Masterchef, the main risk that remains is whether any tokens were pre-minted to the project owner prior to transferring ownership.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED 1000 tokens were pre-minted, and the owner of the token is now the Masterchef.

Issue #22	**`minted` and `burned` variables can overflow causing the variables to misrepresent the actual minted and burned values if extremely large amounts are minted or burned**
Severity	LOW SEVERITY

Location	Lines 697-706
	`function mint(address _to, uint256 _amount) public onlyOwner {`
	`  _mint(_to, _amount);`
	`  minted+=_amount;`
	`}`
	
	`function burn(uint256 _amount) public {`
	`  require (balanceOf(msg.sender)>=_amount);`
	`  _burn(msg.sender, _amount);`
	`  burned+=_amount;`
	`}`

Description	There is no overflow protection on the minted and burned variables which could in theory make them vulnerable to integer overflow, causing these variables to wrap to zero. This is only the case if extremely large amounts are minted or burned and we do not see a way to exploit it.
Recommendation	Consider using SafeMath instead of the current unsafe operators.
Resolution	ACKNOWLEDGED

Issue #23	**`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `updateEmissionRate` and `setFeeAddress` can be made external**
Severity	INFORMATIONAL
Description	The above functions can be changed from `public` to `external`. 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 making these functions `external`.
Resolution	ACKNOWLEDGED

2.3 TokenBurner

The TokenBurner contract collects PolyShield tokens by people/contracts who transfer them to the contract. Anyone can then simply call the burn function on the contract to burn these tokens from the circulating supply. It is a simple and harmless utility contract.

The libraries have not yet been verified.

2.3.1 Issues & Recommendations

Issue #24	**burnTotals can overflow, causing the variables to misrepresent the actual minted and burned values if extremely large amounts are minted or burned**
Severity	LOW SEVERITY

	Lines 23-27
	`function burn() public {`
	`  uint256 tokenBalance = IPolyshield(polyshield).balanceOf(address(this));`
	`  IPolyshield(polyshield).burn(tokenBalance);`
	`  burnTotals[msg.sender]+=tokenBalance;`
	`}`

Description	There is no overflow protection on the minted and burned variables which could in theory make them vulnerable to integer overflow, causing these variables to wrap to zero. This is only the case if extremely large amounts are minted or burned and we do not see a way to exploit it.
Recommendation	Consider using SafeMath instead of the current unsafe operators.
Resolution	ACKNOWLEDGED

Issue #25	`polyshield` 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.
Recommendation	Consider making `polyshield` explicitly `immutable`.
Resolution	ACKNOWLEDGED

Issue #26	`burn` can be made external
Severity	INFORMATIONAL
Description	The `burn` function can be changed from `public` to `external`. 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 making this function `external`.
Resolution	ACKNOWLEDGED

2.4 Timelock

The Timelock contract is a clean fork of Compounder Finance's timelock with the single change that the administrator can be changed at will without going through the timelock delay. Compound Finance's timelock is the most common contract used in DeFi to time lock governance access and is thus compatible with most third-party tools.

Parameter	Value	Description
Delay	12 hours	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	12 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.4.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY