PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Highstreet

10 December 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

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

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

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

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

1 Overview

This report has been prepared for Highstreet Market on the Ethereum 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	Highstreet Market
URL	https://www.highstreet.market/
Platform	Ethereum
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`HighStreetPoolFactory`	`0x03Ce1fd60c31AB8b384725bcb0d8A3A46F87E20f`	MATCH
`HighStreetPoolBase`	`HighStreetPoolBase.sol`	PENDING
`HighStreetFlashPool`	`HighStreetFlashPool.sol`	PENDING
`HighStreetCorePool`	`HIGH Pool: 0xF01e619E8618BFf466877c524a85a5DBe1e36814` `LP Pool: 0x4aeB443981B36B3352e50205469034A8719Ca04E`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	2	-	-
Medium	0	-	-	-
Low	2	1	-	1
Informational	12	12	-	-
Total	**16**	**15**	**-**	**1**

Classification of Issues

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

1.3.1 HighStreetPoolFactory

ID	Severity	Summary	Status
01	LOW	Updating the rewards per block can affect pending rewards retroactively	ACKNOWLEDGED
02	INFO	`updateHighPerBlock` will stop working after 365 years	RESOLVED
03	INFO	`_poolToken` is passed into `getPoolData` but not used	RESOLVED

1.3.2 HighStreetPoolBase

ID	Severity	Summary	Status
04	HIGH	`stake` is vulnerable to re-entrancy, which allows reward amplification on tokens that allow for reentrancy	RESOLVED
05	HIGH	Governance can abuse `totalWeight` to prevent accumulation and collection of rewards, effectively pausing the contract	RESOLVED
06	LOW	Weight calculation biases towards 0 affecting rewards collected by users	RESOLVED
07	INFO	When governance sets a new weight, pending rewards are retroactively changed to use the new weight.	RESOLVED
08	INFO	Usage of `assert` while `require` is more adequate	RESOLVED
09	INFO	Redundant event parameter `_by` in `PoolWeightUpdated`	RESOLVED
10	INFO	Lack of constructor validation: `_initBlock` is not required to be in the future	RESOLVED
11	INFO	`poolToken` is an IERC20 but is stored as an address	RESOLVED
12	INFO	`_isYield` parameter to `_stake` is redundant	RESOLVED
13	INFO	`msg.sender` is unnecessarily cast to `address(msg.sender)`	RESOLVED
14	INFO	Early return causes update logic to not run if `withUpdate` was true and could cause issues or break invariants	RESOLVED
15	INFO	Outdated comments	RESOLVED

1.3.3 HighStreetFlashPool

ID	Severity	Summary	Status
16	INFO	`endBlock` can be made `immutable`	RESOLVED

1.3.4 HighStreetCorePool

No issues found.

2 Findings

2.1 HighStreetPoolFactory

The HighStreetPoolFactory contract is responsible for the creation and management of yield farming pools. It provides a convenient interface for accessing pools, and gives governance and the pools themselves the ability to update their weights.

2.1.1 Privileges

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

- `transferOwnership`
- `renounceOwnership`
- `registerPool`
- `changePoolWeight`

The following functions can be called by the pool:

- `changePoolWeight`

2.1.3 Issues & Recommendations

Issue #01	**Updating the rewards per block can affect pending rewards retroactively**
Severity	LOW SEVERITY
Description	The `updateHighPerBlock` function is called intermittently and if it detects enough blocks have gone by, it will lower all rewards by 3%. When it does this, it fails to pay out any pending rewards at the previous rate, retroactively affecting those rewards by lowering them.
Recommendation	Consider explicitly looping over the pools and paying out rewards for them. Consider adding a flag to prevent this update if the pools grow too big to loop over. ❗ Looping can consume a lot of gas and potentially, when the pools grow bigger, prevent this function from being called successfully. Adding a flag to not update the pools can mitigate this. ❗ Emergency withdrawal facilities can also help mitigate this risk and garner trust from users.
Resolution	ACKNOWLEDGED The client is concerned about the fact that looping might indeed cause the contract to run out of gas and will stay with the current design.

Issue #02	**updateHighPerBlock will stop working after 365 years**
Severity	INFORMATIONAL
Description	When casting down to a lower bit integer, any cast that would cause a loss of information will revert. This means that `blockNumber` can never exceed the maximum value of a uint32 which leaves room for about 365 years of blocks (on BSC) until `updateHighPerBlock` is guaranteed to revert.
Recommendation	Consider using a higher bit datatype like uint256. This issue will also be marked as resolved on the note that the specified timeframe is not an issue.
Resolution	RESOLVED The client has indicated that they will launch exclusively on Ethereum which has much longer block times. If they do move to a different network they will take this issue in consideration.

Issue #03	**_poolToken is passed into getPoolData but not used**
Severity	INFORMATIONAL
Location	Line 178 `address poolToken = IPool(poolAddr).poolToken();`
Description	Within `getPoolData`, `poolToken` is derived on line 178 but the variable `_poolToken`, which has the same value, is passed in already. That makes this computation redundant.
Recommendation	Consider using the already available `_poolToken` instead of calling out to the `poolAddr`.
Resolution	RESOLVED

2.2 HighStreetPoolBase

The HighStreetPoolBase abstract contract is responsible for implementing the common logic that will be used in pools the factory will instantiate. This contract stores a weight that will be set by the deploying factory (see the HighStreetPoolFactory section), and will determine what fraction of the yield is received. It also requires the user to specify a period of time to lock their tokens for yield farming up to one year, though this isn't enforced in this contract, instead inheriting contracts must actually enforce this.

Two planned pools (the core pools) will split the weight 80%/20% the native token (HIGH) pool will have a 20% weight and the pair pool HIGH/Eth will have 80% of the weight.

2.2.1 Privileges

The following functions can be called by the factory:

- `setWeight`

2.2.2 Issues & Recommendations

Issue #04	`stake` is vulnerable to re-entrancy, which allows reward amplification on tokens that allow for reentrancy
Severity	HIGH SEVERITY
Description	The stake function is a vector for re-entrancy. Even though `transferPoolTokenFrom` has a re-entrancy guard, meaning that this function cannot be called again while the previous call is happening, a malicious party is free to reenter in other parts of the code. Specifically the exploit goes as follows: 1. Exploiter calls `stake` 2. Code gives pending rewards to exploiter 3. Code executes until `transferPoolTokenFrom(address(msg.sender), address(this), _amount);` where external contract call occurs and exploiter can inject code 4. Exploiter calls `processRewards` during this injection and receives the pending rewards again 5. Code finishes executing in `_stake` where `subYieldRewards` is only updated at the end Through this exploit, the exploiter can receive their rewards twice if there was a reentrancy vector in the external contract call. Given that the client has added an (insufficient) attempt at protecting themselves against this re-reentrancy, we assume it is a real possibility.

Recommendation	Consider adding re-entrancy guards to all external functions instead of having them at this specific sub-level. Reentrancy guards are not gas expensive and locking down any external function with them is considered good practice within Paladin. Alternatively the code can be carefully redesigned to adhere more to checks-effects-interactions by updating `subYieldRewards` twice. Such a change is however still prone to the risk that some other reentrancy vector would exist. This issue and recommendation also applies for the functionality wihtin `HighStreetFlashPool`.
Resolution	RESOLVED Re-entrancy guards have been added to all relevant functions and the internal guards have furthermore been removed.

Issue #05	**Governance can abuse `totalWeight` to prevent accumulation and collection of rewards, effectively pausing the contract**
Severity	HIGH SEVERITY
Description	In the _sync function to calculate the amount of rewards, a division by the total weight of all pools is done. The total weight can be freely changed by governance (by setting all pools collective weight to 0) and thus prevent _sync from running to completion without reverting. This effectively soft locks the contract and prevents accumulation, distribution and withdrawal of rewards and staked tokens. Given that all withdrawals would be locked under this governance privilege vector, this issue has been marked as high severity. This is based upon the precedence that simple staking contracts should not have the privilege to lock funds indefinitely.
Recommendation	Consider adding emergency withdrawal facilities. The client should be careful to include and reuse all the important logic from the normal withdrawal function except for the reward payout mechanism. The client should also be careful to reset `subYieldRewards` correctly.
Resolution	RESOLVED This situation can now only occur if all pools were flash pools, which Paladin thinks as unlikely. It should be noted that the client did not implement an emergency withdraw function as was recommended.

Issue #06	Weight calculation biases towards 0 affecting rewards collected by users
Severity	LOW SEVERITY
Description	The weight multiplier is 1e6 which is used to increase precision in the calculations involving the new weight of the pool. The problem is 1e6 doesn't garner enough precision to prevent rounding towards 0. This means that rewards can trend towards 0 more often. This effect is amplified if the number of users grows larger and if the supplies of the staking token is very different than that of the rewards token.
Recommendation	Consider using a higher multiplier value like 1e24, testing to ensure there is no overflow.
Resolution	RESOLVED The client now uses 1e24 as the multiplier.

Issue #07	When governance sets a new weight, pending rewards are retroactively changed to use the new weight
Severity	LOW SEVERITY
Description	Governance has the ability to affect the payout of pending rewards retroactively by changing the weight with the `setWeight` function. The `setWeight` function does not `_sync` (calculate and payout rewards) and because of this when the weight is changed, the next call to `_sync` will use this updated value, which will calculate the payout incorrectly and potentially not in favor of the user.
Recommendation	Consider calling `_sync` before updating the weight.
Resolution	RESOLVED Governance can no longer change weights. When a flash pool is disabled, it will sync itself to not affect its own rewards in hindsight. It should be noted that other pools are still affected in hindsight given that the total weight changes which affects all pools. However, the client has voiced concerns with including looping behavior as this has undefined gas usage.

Issue #08	Usage of `assert` while `require` is more adequate
Severity	INFORMATIONAL
Location	Line 433
Description	An assertion is present in this contract that checks if the `stakeWeight` is greater than 0 after calculating the new `stakeWeight`. However, this assertion is not guaranteed to succeed and a simple test case can even be set up to fail the assertion. This test case would involve a token with a 100% transfer-tax, causing `stakeWeight` to become zero.
Recommendation	Consider replacing `assert` with a `require` statement.
Resolution	RESOLVED

Issue #09	Redundant event parameter `_by` in `PoolWeightUpdated`
Severity	INFORMATIONAL
Location	Line 145
Description	`PoolWeightUpdated` can only be triggered by calling `setWeight` which in turn can only be called by the factory. This means that `_by` is redundant as the `_by` address will always be the factory.
Recommendation	Consider removing `_by` in `PoolWeightUpdated` if desired.
Resolution	RESOLVED

Issue #10	**Lack of constructor validation: `_initBlock` is not required to be in the future**
Severity	INFORMATIONAL
Location	Line 162
Description	`_initBlock` is the starting block to calculate rewards from, but is only checked to be greater than 0 and not to be in the future. This means that `lastYieldDistribution` could be 1 for example and the reward calculation would factor in all the blocks from 1 to the current block which would result in astronomical and unintentional yield.
Recommendation	Consider requiring `_initBlock` to be in the future.
Resolution	RESOLVED

Issue #11	**`poolToken` is an IERC20 but is stored as an address**
Severity	INFORMATIONAL
Description	The `poolToken` variable is currently stored as the address type. `poolToken` could be more succinctly stored as an IERC20 instead of an address, this would remove the need to cast it to IERC20 at every usage site.
Recommendation	Consider changing the type of `poolToken` to IERC20.
Resolution	RESOLVED The client prefers the semantics of storing it as an address. This is a purely syntax issue so there's no functional advantage to either approach.

Issue #12	`_isYield` parameter to `_stake` is redundant
Severity	INFORMATIONAL
Description	`_stake` takes a parameter, `_isYield` to signify if the deposit yields or not. However in practice `_stake` is only called with false and yielding deposits are created without ever calling into `_stake`. This makes the `_isYield` parameter unnecessary and redundant.
Recommendation	Consider removing the unused `_isYield` parameter from `_stake` if desired.
Resolution	RESOLVED

Issue #13	`msg.sender` is unnecessarily cast to `address(msg.sender)`
Severity	INFORMATIONAL
Description	`msg.sender` is cast to `address(msg.sender)` throughout the contract when used with `pool.lpToken.safeTransfer()`. This is unnecessary.
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with msg.sender.
Resolution	RESOLVED

Issue #14	**Early return causes update logic to not run if `withUpdate` was true and could cause issues or break invariants**
Severity	INFORMATIONAL
Location	Line 589 `If (pendingYield == 0) return 0;`
Description	The `_processRewards` function contains an early return. While we could not find any direct consequences of this, `subYieldRewards` will never be updated if `pendingYield` is 0, this could break invariants within the contract and is especially a risk if the contract gets upgraded or forked. Early returns while actions within the function might be expected have been the cause of many exploits. ❗ `_processVaultRewards` in the HighStreetCorePool has similar behavior. In order to not bloat the report, the issue has not been duplicated but it should be considered for both contracts.
Recommendation	Consider still updating `subYieldRewards` if this is part of `_processRewards`' invariant.
Resolution	RESOLVED The client has indicated that they will keep this behavior in mind if they ever decide to append on this function in new deployments.

Issue #15	**Outdated comments**
Severity	INFORMATIONAL
Description	Within HighStreetPoolBase and HighStreetFlashPool, comments of the form mentioned below are included — these comments are incorrect as a normal call is made. `// just delegate call to the target`
Recommendation	Consider removing these comments.
Resolution	RESOLVED

2.3 HighStreetFlashPool

The HighStreetFlashPool contract is not a core pool (permanent pool) but a temporary one. It allows you to stake for exactly a 1 year period, however your tokens are never locked and can be unstaked at any time. It inherits HighStreetPoolBase.

It should be noted that the contract documentation says that this contract allows staking for an exact 1 year period while in reality people can stake and earn rewards until the configured `endBlock`, they could also still stake one day before that block is reached for example.

2.3.1 Privileges

The following functions can be called by the factory:

- `setWeight`

2.3.2 Issues & Recommendations

Issue #16	**endBlock 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

2.4 HighStreetCorePool

The HighStreetCorePool contract represents a pool which allows for dual rewards with a part of the rewards coming from the vault. Similar to the Flash Pool, this contract inherits all properties and therefore also all issues of the PoolBase.

Different from the Flash Pool however, is that users can lock themselves out of withdrawing for up to a year: their rewards will be amplified linearly from 100% to 200% depending on how long they commit to not withdrawing. The reward weight is therefore 100% in case the user locks up for 0 days and 200% if they lockup for a year. While locked up, withdrawals are not allowed in any form while harvests can still occur. Furthermore, other pools can stake into this pool as a compounding vehicle to compound HIGH tokens on harvests.

2.4.1 Privileges

The following functions can be called by the other pools:

- `stakeAsPool`

See HighStreetPoolBase for the other privileged positions.

2.4.2 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY