PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Avalaunch (Scope Extension)

24 March 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 Avalaunch on the Avalanche network. This round of audit is an extension to the scope of the original audit. 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.

This is a follow-up report of our first Avalaunch audit. The Avalaunch team has made several adjustments to their AvalaunchSale contract which are covered within this audit report. All changes to the AvalaunchSale up to commit hash 672f49ccc180fd6af812db71ed8e744a802ce4a1 are covered within this report.

1.1 Summary

Project Name	Avalaunch
URL	https://avalaunch.app/
Platform	Avalanche
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
AvalaunchCollateral	Proxy 0x7E5f6AB97EEf4f28900Dc0F713EB99d3c077BBda Implementation 0xa95da4598d509f621f45d8b97283928bd0815ca4	MATCH
AvalaunchSale	Proxy 0x0450cfd41a9bba5349f50a75043d69e8d96f2f9e Implementation 0x0a1a9eb0d984f1c194c85bace2070724101272e3	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	3	2	-	1
Medium	0	-	-	-
Low	2	2	-	-
Informational	7	3	1	3
Total	**12**	**7**	**1**	**4**

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 AvalaunchCollateral

ID	Severity	Summary	Status
01	HIGH	`userBalance` is not reduced on `boostParticipation`	RESOLVED
02	HIGH	`permitSignature` does not sign sufficient values which could allow governance to take more user-deposited AVAX from their collateral than the user might expect	ACKNOWLEDGED
03	INFO	SafeMath can only be used for `uint256`	RESOLVED
04	INFO	Lack of indexing for event parameters	RESOLVED
05	INFO	Tyopographical errors	ACKNOWLEDGED
06	INFO	Gas optimizations	ACKNOWLEDGED

1.3.2 AvalaunchSale

ID	Severity	Summary	Status
07	HIGH	`boostedAmountBought` is unused throughout the contract	RESOLVED
08	LOW	There is no cap of token to sell within the boost round	RESOLVED
09	LOW	The `amountOfTokensBuying` limit during the boost round is wrongly based on staking round	RESOLVED
10	INFO	The function `setAndSupportDexalotPortfolio` should not be settable after the gate is closed	ACKNOWLEDGED
11	INFO	Lack of validation	RESOLVED
12	INFO	Typographical errors	PARTIAL

2 Findings

2.1 AvalaunchCollateral

AvalaunchCollateral is a contract where users can deposit their AVAX to be able to automatically participate in sales once the user has given permission to Avalaunch through a gassless, private-key signed message. The user will then participate in this launch using the AVAX sent by them to the AvalaunchCollateral contract beforehand. A small fee will be paid to the moderator.

This contract also includes the option for users to boost their participation by buying more tokens than they could have in the round they participated. The execution of this "boost" functionality follows a similar automated flow.

The user is able to withdraw their unused AVAX at anytime.

2.1.1 Privileges

The following functions can be called by the owner:

- `autoParticipate`
- `boostParticipation`
- `setModerator`
- `approveSale`
- `addAdmin`
- `removeAdmin`

2.1.2 Issues & Recommendations

Issue #01	**userBalance is not reduced on boostParticipation**
Severity	HIGH SEVERITY
Location	Line 191-224 `function boostParticipation(`
Description	`userBalance` is not reduced on `boostPaticipation`, which means that after a user boosts their participation by providing AVAX, this user can still withdraw said AVAX from the contract using `withdrawCollateral`.
Recommendation	Consider reducing `userBalance` on `boostPaticipation`.
Resolution	RESOLVED `userBalance` is now properly decreased.

Issue #02	**`permitSignature` does not sign sufficient values which could allow governance to take more user-deposited AVAX from their collateral than the user might expect**
Severity	HIGH SEVERITY
Description	The message users sign currently only takes a list of hashed strings and `thesaleContract` address. This means that the different amounts, such as `amountAVAX`, `amount` and `amountXavaToBurn` and the `roundId` are not checked. Any value could be used and the signature scheme would still be right. Specifically, this means that there's no way for the user to make sure that Avalaunch does not take undesirable values from their collateral, staked XAVA, etc.
Recommendation	Consider signing all the important values.
Resolution	ACKNOWLEDGED The client has indicated that this is a governance privilege that they are willing to accept. We've indicated to the client to tread carefully if the admin private keys are uploaded as any one private key can potentially burn the staked XAVA of a participator.

Issue #03	**SafeMath can only be used for `uint256`**
Severity	INFORMATIONAL
Location	Line 11 `using SafeMath for *;`
Description	Presently the contract applies the SafeMath functionality to all types. However, this functionality is only supposed to be used with `uint256`. Using wildcards for such functionality can be confusing for third-parties and also lead to development error, where co-developers might expect SafeMath to also correctly function on other integer types.
Recommendation	Consider using SafeMath only for `uint256`.
Resolution	RESOLVED

Issue #04	Lack of indexing for event parameters
Severity	INFORMATIONAL

Location	Line 42~
	`event DepositedCollateral(address wallet, uint256 amountDeposited, uint256 timestamp);`
	`event WithdrawnCollateral(address wallet, uint256 amountWithdrawn, uint256 timestamp);`
	`event FeeTaken(address sale, uint256 participationAmount, uint256 feeAmount, string action);`
	`event ApprovedSale(address sale);`

Description	Essential identifying parameters within events should be marked as indexed. This allows for off-chain components to filter events based on these parameters.
Recommendation	Consider marking addresses as indexed.
Resolution	RESOLVED

Issue #05	Typographical errors
Severity	INFORMATIONAL

Description	The `chainId` comment parameter is missing on `initialize` function.
	
	Consider adding a comment to keep consistency in the contract.
	
	It should be noted that the `chainId` can actually be retrieved in Solidity and it does not need to be a parameter with the following code:
	`uint256 chainId;`
	`assembly {`
	`chainId := chainid()`
	`}`
	
	The key type of `isSignatureUsed` can be of type `bytes32` to make the actual type more explicit.
	
	The modifiers can be consistently positioned behind `external` and `public`. This is not done within `setModerator` for example, making the function modifier order inconsistent over the contract.

Recommendation	Consider fixing the typographical errors.
Resolution	ACKNOWLEDGED

Issue #06	Gas optimizations
Severity	INFORMATIONAL
Description	Various typehashes are marked as constant functions within the variables section of the codebase (eg. `BOOST_TYPEHASH`). Due to the way the Solidity compiler works, these are converted into pure functions which waste gas executing `abi.encodePacked` and `keccak256`.
Recommendation	Consider hard-coding the type hashes. Running a quick test on this, it seems to save about 500 gas whenever the type hash is fetched. This issue will also be resolved on the note that Avalaunch prefers readability over gas-optimality.
Resolution	ACKNOWLEDGED

2.2 AvalaunchSale

The AvalaunchSale contract is a contract which is deployed by the SalesFactory for every project which has its tokens sold on Avalaunch. It is the contract which users send AVAX to, in order to eventually withdraw the launch project's tokens.

There's a registration fee for every sale hat users participate in. Users have to register before the sale starts. If they do participate, they receive this fee back. If users decide not to purchase tokens after they have registered, the registration fee goes to Avalaunch. It should be noted that users are solely able to participate with a valid off-chain signature from the Avalaunch website. If the website were to go offline, they might accidentally lose their registration fee. We hope and expect Avalaunch to reimburse users in this unlikely scenario.

The contract is not designed to distribute fee on transfer tokens due to the `depositTokens` function which does not account for them. The team should remember to always exclude the sale from any potential transfer taxes. The team should also keep in mind that `tokenPriceInAvax` has 18 decimals of precision.

Users can register for a specific allocation round. If the user registers for the staking round, their stake in the AllocationStaking contract will be locked until the sale has ended. Each sale has one round which is the allocation round. If the user participates in this round, their locked allocation will be partially redistributed within the AllocationStaking contract.

Finally, the contract contains logic to have multiple vesting cliffs of the purchased tokens (for example once every month for 12 months).

The client has made several adjustments to this contract since our previous audit, and all changes up to commit hash `672f49ccc180fd6af812db71ed8e744a802ce4a1` are covered within this report.

2.2.1 Issues & Recommendations

Issue #07	**boostedAmountBought is unused throughout the contract**
Severity	HIGH SEVERITY
Description	This update includes the possibility for users to buy a "boosted amount" of tokens. Currently the `boostAmount` is only stored but never used. The `boostedAmountBought` should probably be added to the different vesting portions or transferred to the user using another way.
Recommendation	Consider using `boostedAmountBought` within the contract to send that amount to users.
Resolution	RESOLVED The user's `amountBought` is now incremented on boosted purchases.

Issue #08	**There is no cap of token to sell within the boost round**
Severity	LOW SEVERITY
Description	Within `boostParticipation`, there is no cap on the amount to sell to users. The value should probably be checked to be lower than that is left inside the contract. Presently, this is done within other locations of the code which turns this into an inconsistency.
Recommendation	Consider adding a sale cap and implementing the following code: `require( amount <= sale.amountOfTokensToSell.sub(sale.totalTokensSold), "Trying to buy more than contract has." );`
Resolution	RESOLVED The recommended code has been implemented.

Issue #09	**The `amountOfTokensBuying` limit during the boost round is wrongly based on staking round**
Severity	LOW SEVERITY

Location	Lines 747~750
	`require(`
	`  amountOfTokensBuying <=`
	`roundIdToRound[stakingRoundId].maxParticipation,`
	`  "Overflowing maximal participation for this round."`
	`);`

Description	The `maxParticipation` of the `boostedRound` is wrongly checked to that of the `stakingRound`.
Recommendation	Consider checking the `boostingRoundId` instead of the `stakingRoundId`.
Resolution	RESOLVED

Issue #10	**The function `setAndSupportDexalotPortfolio` should not be settable after the gate is closed**
Severity	INFORMATIONAL

Location	Lines 307~313
	`function setAndSupportDexalotPortfolio(`
	`  address _dexalotPortfolio,`
	`  uint256 _dexalotUnlockTime`
	`)`
	`external`
	`onlyAdmin`
	`{`

Description	The function `setAndSupportDexalotPortfolio` should not be settable after the gate is closed.
Recommendation	Consider adding the `onlyGateOpen` modifier to the function.
Resolution	ACKNOWLEDGED

Issue #11	Lack of validation
Severity	INFORMATIONAL

Description	The `_colateral` address is presently not checked against `address(0)`, while the other are.
	
	Lines 176~187
	`function initialize(`
	`  address _admin,`
	`  address _allocationStaking,`
	`  address _collateral`
	`) public initializer {`
	`  require(_admin != address(0));`
	`  require(_allocationStaking != address(0));`
	`  admin = IAdmin(_admin);`
	`  factory = ISalesFactory(msg.sender);`
	`  allocationStakingContract =`
	`IAllocationStaking(_allocationStaking);`
	`  collateral = ICollateral(_collateral);`
	`}`
	
	The parameters `stakingRoundId` and `boosterRoundId` of the function `setSaleParams` lacks validation.
	
	Furthermore, the `stakingRoundId` and `boosterRoundId` are set before the rounds are initialized using `setRounds`, which is considered as a design flaw. Consider setting them inside `setRounds` or assert that they exist when setting rounds by checking that `roundIds` array is long enough.

Recommendation	Consider checking that the `_collateral` address is not `address(0)` to promote consistency. Consider also implementing the recommendations with regards to the `roundIds` as described above.
Resolution	RESOLVED

Issue #12	Typographical errors
Severity	INFORMATIONAL

Description	The different parameters can be directly cast to their respective types.
	
	Lines 176~187
	`function initialize(`
	`  address _admin,`
	`  address _allocationStaking,`
	`  address _collateral`
	`) public initializer {`
	`  require(_admin != address(0));`
	`  require(_allocationStaking != address(0));`
	`  admin = IAdmin(_admin);`
	`  factory = ISalesFactory(msg.sender);`
	`  allocationStakingContract = IAllocationStaking(_allocationStaking);`
	`  collateral = ICollateral(_collateral);`
	`}`
	
	Consider casting them directly to their respective types.
	
	The check limiting the amount bought should be greater or equal than amount rather than strictly greater.
	
	Line 745
	`require(amountOfTokensBuying < amount, "Trying to buy more than allowed.");`
	
	Consider changing it to `amountOfTokenBuying <= amount`.
	
	The parameter `_dexalotPortfolio` of the function `setAndSupportDexalotPortfolio` can be cast directly to `IDexalotPortfolio`.

Recommendation	Consider fixing the typographical errors.
Resolution	PARTIALLY RESOLVED The `initialize` and `setAndSupportDexalotPortfolio` function signatures have been left as is.


PALADIN
BLOCKCHAIN SECURITY