



**PALADIN**  
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

For Avalaunch (Sale v2)

23 January 2023



[paladinsec.co](https://paladinsec.co)



[info@paladinsec.co](mailto:info@paladinsec.co)

# Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 AvalaunchSaleV2	6
1.3.2 SalesFactory	7
1.3.3 AvalaunchMarketplace	7
2 Findings	8
2.1 AvalaunchSaleV2	8
2.1.1 Privileged Functions	10
2.1.2 Issues & Recommendations	11
2.2 SalesFactory	28
2.2.1 Privileged Functions	28
2.2.2 Issues & Recommendations	29
2.3 AvalaunchMarketplace	33
2.3.1 Privileged Functions	33
2.3.2 Issues & Recommendations	34



# 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 advocacy 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's Sale v2 contracts on the Avalanche 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

<b>Project Name</b>	Avalaunch
<b>URL</b>	<a href="https://avalaunch.app/">https://avalaunch.app/</a>
<b>Platform</b>	Avalanche
<b>Language</b>	Solidity

## 1.2 Contracts Assessed

Name	Contract	Live Code Match
AvalaunchSaleV2		
SalesFactory		
AvalaunchMarketplace		

## 1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	2	2	-	-
● Medium	3	2	-	1
● Low	13	9	1	3
● Informational	8	4	-	4
Total	26	17	1	8

### 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 AvalaunchSaleV2

ID	Severity	Summary	Status
01	HIGH	The <code>_participate</code> function refunds the registration fee as <code>sale.token</code> instead of AVAX, breaking the contract completely	✓ RESOLVED
02	MEDIUM	The contract does not work with tokens that have a fee on transfer	✓ RESOLVED
03	MEDIUM	Governance privileges: The Avalaunch team has exceptional control over the functioning of the protocol	ACKNOWLEDGED
04	LOW	Missing length validation for <code>setVestingParams</code>	ACKNOWLEDGED
05	LOW	Signature scheme is suboptimal which means it might lead to replays on chain forks	ACKNOWLEDGED
06	LOW	Checks-effects-interactions pattern is not adhered to	✓ RESOLVED
07	LOW	<code>registerForSale</code> can be called while the sale is not yet created and even if <code>sale.saleEnd</code> has been exceeded	✓ RESOLVED
08	LOW	Users might not be able to exhaust their full allowance	ACKNOWLEDGED
09	LOW	<code>saleEnd</code> cannot be reconfigured if no portions have been configured yet	✓ RESOLVED
10	LOW	Dexalot logic can severely malfunction for tokens without a symbol	✓ RESOLVED
11	INFO	Gas optimizations	✓ RESOLVED
12	INFO	Validation: Contract does not prevent moderator from accidentally depositing tokens twice	✓ RESOLVED
13	INFO	Validation: Lack of phase validation for different participation methods	PARTIAL
14	INFO	AVAX can get stuck in the implementation contract	✓ RESOLVED
15	INFO	Typographical issues	PARTIAL

## 1.3.2 SalesFactory

ID	Severity	Summary	Status
16	LOW	getAllSales has an incorrect input validation which causes the initial requirement to pass even though the endIndex is out of range	PARTIAL
17	LOW	Sales moderator is a fixed wallet which cannot be changed	RESOLVED
18	LOW	Checks-effects-interactions pattern is not respected	RESOLVED
19	INFO	Typographical errors	PARTIAL
20	INFO	Gas optimizations	PARTIAL

## 1.3.3 AvalaunchMarketplace

ID	Severity	Summary	Status
21	HIGH	Signature lacks replay protection	RESOLVED
22	MEDIUM	Governance risk: Admin can frontrun any purchase with an increase of the fee parameters	RESOLVED
23	LOW	There is no way to remove sale contracts	RESOLVED
24	LOW	Checks-effects-interactions pattern is not adhered to	RESOLVED
25	LOW	Contract deployment does not disable the initializer	RESOLVED
26	INFO	Typographical errors	RESOLVED

## 2 Findings

---

### 2.1 AvalaunchSaleV2

AvalaunchSaleV2 is the updated main presale contract on the Avalaunch launchpad. This contract has an abundance of features making Avalaunch sales some of the most interesting mechanisms out there.

From a high level, sales have the following phases:

- Idle (configuration and marketplace phase)
- Registration
- Participation
  - Validator
  - Staking
  - Booster

During the idle phase, the sale is configured by the Avalaunch administrators. Examples of configurable parameters are: the token which will be sold, amount that will be sold, how the unlocks will occur, and more.

Once configured, admins can enable the registration phase which allows users to sign up to the sale provided the administrators gave them permission through an off-chain signature. Registration incurs a fee, however, which will be paid back to the users.

After the registration phase, the participation phase begins, where users can buy tokens using AVAX tokens. During this phase, the tokens that have been bought will be locked and cannot be claimed until the vesting period is over.



The contract also has a feature where tokens can be deposited into a platform called Dexalot.

Throughout the different phases, the administrators can also shift the sale end date and change the parameters related to the way the tokens can be bought.

During the vesting period, purchased tokens are divided into several portions. Each portion has a specific unlocking time, and a percentage of tokens that will be unlocked at that time. This means that the tokens are locked for a specific period of time and will be gradually released over that period.

Purchased tokens can be withdrawn or listed on the marketplace depending on their portion state. The portion state can be "Available", "Withdrawn", "WithdrawnToDexalot", "OnMarket", or "Sold".

When a portion is in the "Available" state, it means that the tokens have not been withdrawn or listed on the marketplace yet. The tokens can be withdrawn by the user at any time after the unlocking period has passed.

When a portion is in the "Withdrawn" state, it means that the tokens have been withdrawn and can be used.

When a portion is in the "WithdrawnToDexalot" state, it means that the tokens have been withdrawn and deposited on the Dexalot platform, which can also be used to buy the tokens.

When a portion is in the "OnMarket" state, it means that the tokens have been listed for sale on a marketplace and can be bought by other users.

When a portion is in the "Sold" state, it means that the tokens have been bought by another user on the marketplace.



In summary, the vesting portions feature allows tokens to be bought but will be locked for a specific period of time, and these locked tokens are divided into



several portions, each with its own unlocking time and percentage of tokens that will be unlocked at that time. The user can either withdraw or list the tokens for sale on a marketplace, depending on the portion state.

## 2.1.1 Privileged Functions

- `setVestingParams [ admins ]`
- `shiftVestingUnlockTimes [ admins ]`
- `setSaleParams [ admins ]`
- `shiftSaleEnd [ admins ]`
- `setDexalotParameters [ admins ]`
- `shiftDexalotUnlockTime [ admins ]`
- `setSaleToken [ admins ]`
- `updateTokenPriceInAVAX [ admins ]`
- `overrideTokenPrice [ admins ]`
- `withdrawRegistrationFees [ admins ]`
- `withdrawUnusedFunds [ admins ]`
- `removeStuckTokens [ admins ]`
- `changePhase [ admins ]`
- `activateLock [ admins ]`
- `depositTokens [ moderator ]`
- `withdrawEarningsAndLeftover [ moderator ]`
- `autoParticipate [ collateral contract ]`
- `boostParticipation [ collateral contract ]`
- `transferPortions [ marketplace contract ]`

## 2.1.2 Issues & Recommendations

<b>Issue #01</b>	<b>The <code>_participate</code> function refunds the registration fee as <code>sale.token</code> instead of AVAX, breaking the contract completely</b>
<b>Severity</b>	 HIGH SEVERITY
<b>Description</b>	<p>Currently, the <code>_participate</code> function transfers the sale token back to the user instead of AVAX for the registration refund:</p> <p><u>Line 580</u></p> <pre>sale.token.safeTransfer(user, registrationDepositAVAX);</pre> <p>This is incorrect, as the fee was paid in AVAX therefore the refunded amount should be in AVAX. By sending back the sale token, the contract misbehaves and this refund could be extremely destructive in case the sale token has a small supply (eg. 1 "AVAX").</p>
<b>Recommendation</b>	Consider transferring back AVAX instead. Also keep in mind checks-effects-interactions.
<b>Resolution</b>	 RESOLVED

<b>Issue #02</b>	<b>The contract does not work with tokens that have a fee on transfer</b>
<b>Severity</b>	 MEDIUM SEVERITY
<b>Description</b>	Currently, the <code>depositTokens</code> function transfers the desired amount from the admin to the contract. However, if the token has a fee on transfer, the contract will receive less tokens than expected, which will cause <code>withdrawMultiplePortions</code> to revert for the last withdrawer(s).
<b>Recommendation</b>	Consider either whitelisting the sale contract or optimizing the <code>depositTokens</code> function to only set <code>sale.tokensDeposited</code> to <code>true</code> if the contract actually received enough tokens. Alternatively, the admin can just send the remaining amount directly to the contract.
<b>Resolution</b>	 RESOLVED The Avalaunch team does intentionally not support tokens with a fee on transfer for now. This feature might be implemented in a future upgrade if desired.

**Issue #03****Governance privileges: The Avalaunch team has exceptional control over the functioning of the protocol****Severity** MEDIUM SEVERITY**Description**

The contract contains a number of governance privileges which allows the owner to potentially carry out malicious actions. To keep the report size reasonable, we have enumerated all governance privileges in a single issue:

Line 224

```
function shiftVestingUnlockTimes(uint256 timeToShift)
external onlyAdmin
```

The admin can shift the unlock times so they can never be reached. The same issue applies to dexalotUnlocktime.

Line 838


```
function changePhase(Phases _phase) external onlyAdmin
```

The admin can switch phases freely within the active sale timeframe. However, due to the fact that the admin can also switch the sale time via shiftSaleEnd, they essentially have full control over the phases.

While we assess the Avalaunch team to be highly trustworthy and we do not expect any malicious behavior, centralization risks cannot be ignored fully, especially and specifically since several hot wallets will have access to these privileges.

**Recommendation**

Consider openly communicating all future changes with the community, consider carefully placing all admin keys behind secure key management systems (eg. AWS solutions), consider carefully managing risk (eg. only adding liquidity to Avalaunch sales after the sale completed successfully), and consider using RBAC instead of the general single admin role.

**Resolution** ACKNOWLEDGED

The Avalaunch team will move towards a safer governance solution in the future and has reiterated how they take governance security seriously.


**Issue #04****Missing length validation for setVestingParams****Severity** LOW SEVERITY**Description**

setVestingParams allows for an arbitrary length of the provided parameters, and ensures they have the same length. Within this function, numberOfVestedPortions becomes the length of these inputs.

The parameter is often used to loop over, while the loop in the setter function might not run out of gas, it can become an issue in other functions that consume more gas or in other contracts that might loop twice over such parameters.


**Recommendation**

Consider setting a reasonable upper limit for the length of the input parameters.

**Resolution** ACKNOWLEDGED

The Avalaunch team will consider adding a length limit in the future.



**Issue #05****Signature scheme is suboptimal which means it might lead to replays on chain forks****Severity** LOW SEVERITY**Description**

Avalaunch uses an ad-hoc signature scheme to authorize transactions by the Avalaunch team. This is suboptimal as there have been made standards for more secure signing of transactions which are more robust.

Additionally, Avalaunch uses `abi.encodePacked` for all of their signature datas. If the data contains two variable length fields, this could lead to collisions. Presently such collisions do not appear to be possible as all the signature datas with 2 variable fields have the second field (the string) as an expected fixed length.



Currently, the only replay protection for the signature is `sigExpTimestamp`. This means that the provided transaction can be executed arbitrarily often until the timestamp is reached.

**Recommendation**



Consider using EIP-712: <https://eips.ethereum.org/EIPS/eip-712>. Consider using `abi.encode` instead of `abi.encodePacked`.



**Resolution** ACKNOWLEDGED



The Avalaunch team will consider implementing the EIP-712 standard in the future. `abi.encodePacked` is kept for gas reasons as there are no collisions possible with the current setup.

<b>Issue #06</b>	<b>Checks-effects-interactions pattern is not adhered to</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>Currently, the contract does not always adhere to the checks-effects-interactions pattern:</p> <p><u>Line 378-388</u></p> <pre>if (phaseId == uint8(Phases.Staking)) {  allocationStaking.setTokensUnlockTime( 0, msg.sender, sale.saleEnd );  } // Increment number of registered users numberOfRegistrants++; // Increase earnings from registration fees registrationFees += msg.value;</pre> <p><u>Line 782-789</u></p> <pre>function withdrawRegistrationFees() external onlyAdmin { require(block.timestamp &gt; sale.saleEnd, "Sale isn't over."); require(registrationFees &gt; 0, "No fees accumulated."); // Transfer AVAX to the admin wallet safeTransferAVAX(msg.sender, registrationFees); // Set registration fees to zero registrationFees = 0; }</pre> <p>Especially for the latter code snippet, the admin can drain the contract via reentrancy, however, we do not see this as an issue because the admin can take out any unused funds either way.</p> <p>The redistribution of Xava is also not written in CEI on line 552.</p>
<b>Recommendation</b>	Consider adhering to the checks-effects-interactions pattern.
<b>Resolution</b>	 RESOLVED





<b>Issue #07</b>	<b>registerForSale can be called while the sale is not yet created and even if sale.saleEnd has been exceeded</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>Currently, the only limitation besides the signature is the requirement for the current phase to be in the Registration phase. However, if the admin accidentally sets the phase to Registration before the sale was created, all users can register themselves for free.</p> <p>_participate only ensures that the contract state is in a valid phase. However, if sale.saleEnd has been exceeded, users can still participate in a sale.</p>
<b>Recommendation</b>	<p>Consider adding an additional requirement require(sale.isCreated) to ensure that the sale is actually created.</p> <p>Consider checking saleEnd if desired.</p>
<b>Resolution</b>	 RESOLVED <p>The sale isCreated flag is checked now.</p>

<b>Issue #08</b>	<b>Users might not be able to exhaust their full allowance</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>_participate allows only for one participation in the Validation or Staking phases.</p> <p>If a user accidentally sends msg.value (AVAX) which does not exhaust their granted allowance amount, they will get effectively locked out from any further participation.</p>
<b>Recommendation</b>	Since this issue needs quite some refactoring of the _participate logic, we recommend openly communicating with users that only one regular participation is possible.
<b>Resolution</b>	 ACKNOWLEDGED

<b>Issue #09</b>	<b>saleEnd cannot be reconfigured if no portions have been configured yet</b>
<b>Severity</b>	 LOW SEVERITY
<b>Location</b>	<u>Line 289</u> <pre>require(sale.saleEnd &lt; vestingPortionsUnlockTime[0], "Sale end crossing vesting unlock times.");</pre>
<b>Description</b>	The contract defines an administrative shiftSaleEnd function which allows the administrator to move the end of the sale backwards. However, this is impossible to do before configuring the vesting portions due to the first portion being accessed. If this portion does not exist, it reverts implicitly due to an out of range error.
<b>Recommendation</b>	Consider adjusting the requirement to be robust. Alternatively consider acknowledging that you cannot adjust the sale end before setting the vesting portions.
<b>Resolution</b>	 RESOLVED



<b>Issue #10</b>	<b>Dexalot logic can severely malfunction for tokens without a symbol</b>
<b>Severity</b>	 LOW SEVERITY
<b>Location</b>	<u>Line 905</u> <code>_symbol := mload(add(symbol, 32))</code>
<b>Description</b>	<p>The contract uses a shortcut to load in the first 32 bytes of the symbol as the dexalot ID. However, since this shortcut uses assembly, it is actually quite brittle.</p> <p>In this case, this actually poses an issue when the symbol is zero bytes long as random memory is accessed and provided as the symbol. This scenario would cause the dexalot functionality to completely break and in the worst case unexpectedly pull some other token from the user's wallet.</p>
<b>Recommendation</b>	Consider adjusting the requirement to be robust. Alternatively, consider acknowledging that you cannot adjust the sale end before setting the vesting portions.
<b>Resolution</b>	 RESOLVED <p>These tokens are explicitly no longer permitted through a length requirement.</p>

**Description**

We have consolidated the sections which can be further optimized for gas usage below.

Line 207, 228, 561

```
for (uint256 i = 0; i < numberOfVestedPortions; i++) {  
    require(portionId < numberOfVestedPortions, "Invalid portion  
id.");  
}
```

numberOfVestedPortions is fetched from storage within each loop iteration.

Line 274-278

```
emit SaleCreated( sale.tokenPriceInAVAX,  
sale.amountOfTokensToSell, sale.saleEnd );
```

Some gas can be saved by passing the function parameters instead of accessing the storage variables.

Line 353, 882

```
bytes memory signature,  
function verifySignature(bytes32 hash, bytes memory  
signature) internal view {
```

This signature can be provided as calldata to save on gas.

Line 524

```
(msg.value).mul(uint(10) **  
IERC20Metadata(address(sale.token)).decimals()).div(sale.token  
PriceInAVAX);
```

The sale token decimals can be cached to save on gas.

---

Line 563

```
lastPercent = vestingPercentPerPortion[i];
```

This is already fetched. The if statement could be refactored out by simply caching lastPercent and portionVestingPrecision and then always doing the mul-div. This would make the logic slightly more simple as there is no longer an if-branch occurring.

Line 870-871

```
portionAmounts: _emptyUint256,  
portionStates: _emptyPortionStates
```

Consider initializing these arrays through pure functions instead as we believe right now there is a lot of storage access at these lines.

---

<b>Recommendation</b>	Consider implementing the recommendations.
-----------------------	--

---

<b>Resolution</b>	 <b>RESOLVED</b>
-------------------	---

---

---

<b>Issue #12</b>	<b>Validation: Contract does not prevent moderator from accidentally depositing tokens twice</b>
------------------	--

---

<b>Severity</b>	 <b>INFORMATIONAL</b>
-----------------	--

---

<b>Location</b>	<u>Line 437</u> <code>sale.tokensDeposited = true;</code>
-----------------	--

---

<b>Description</b>	depositTokens sets a boolean to true, however, this boolean is never checked to be false. This allows for the moderator to call this function multiple times, and they can potentially accidentally deposit more tokens than is necessary.
--------------------	--

---

<b>Recommendation</b>	Consider checking this boolean to be false or acknowledging the issue. In case its desired to add more tokens to the contract, they can always be transferred directly.
-----------------------	---

---

<b>Resolution</b>	 <b>RESOLVED</b>
-------------------	---

---

**Issue #13****Validation: Lack of phase validation for different participation methods****Severity** INFORMATIONAL**Description**

The different methods of participation are meant to be used with different phases (eg. the booster phase or everything but the booster phase). However, this is not validated to be correct other than with the admin signature.

Additionally, on a side-note, the checks within the participation function are scattered all over the function which makes the code extremely messy. There are many `if` statements to check whether the participation is a booster, and it would be cleaner to move all checks (requirements) to the top of the function whenever possible (this is not possible for checks on the amount of tokens purchased, of course).

Finally, we wonder about validating `amountXavaToBurn`, as it should probably remain zero for anything but the booster and staking phases (any other number does not do anything but might be confusing).

**Recommendation**

Consider whether subsequent validation is desired. Consider moving the checks to the top of the function to be more aligned with checks-effects-interactions.

**Resolution** PARTIALLY RESOLVED

A logic change within `redistributeXava` was introduced which was then reversed in the latest commit.

**Issue #14****AVAX can get stuck in the implementation contract****Severity** INFORMATIONAL**Description**

Currently, the implementation contract does not call the function `_disableInitializers` during deployment. A malicious attacker can therefore initialize the implementation directly, setting their own address as admin and potentially retrieve any ether which gets stuck within the contract when other users accidentally interact directly with the implementation.

**Recommendation**

Consider calling `_disableInitializers` within the constructor.

**Resolution** RESOLVED

The implementation is now disabled which prevents anyone from taking the AVAX.



**Description**

We have consolidated the typographical errors into a single issue to keep the report brief and readable.

Line 5

```
import "../interfaces/ISalesFactory.sol";
```

The import is unused, though we assume this (the factory variable) might be for UI purposes.

Lines 11&12

```
import "@openzeppelin/contracts/cryptography/ECDSA.sol";  
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
```

The contract uses non-upgradeable libraries. We expect this to be all right however as libraries do not contain state constructors or state.

Line 24

```
ISalesFactory public factory;
```

This line appears to be unused (though it is not strictly an issue, as this might be useful for seeing in the explorer).

Line 159-165

```
function initialize( address _admin, address  
_allocationStaking, address _collateral, address  
_marketplace, address _moderator )
```

The provided parameters can be directly cast as their corresponding type.

Line 209

```
// Each portion unlock time must be latter than previous  
  
"latter" should be "later".
```



---

Line 245

address \_token,

This parameter can be provided as IERC20 directly.

Line 300

address \_dexalotPortfolio,

The parameter can be directly cast with IDexalotPortfolio.

Line 309-315

```
require( _dexalotPortfolio != address(0) &&
_dexalotUnlockTime > sale.saleEnd && _dexalotUnlockTime <=
vestingPortionsUnlockTime[0] && vestingPortionsUnlockTime[0]
> 0 && sale.saleEnd > 0 );
```

The require statement should return a clear error string.

Line 325

```
require(block.timestamp < dexalotUnlockTime &&
shiftedDexalotUnlockTime <= vestingPortionsUnlockTime[0]);
```

The require statement should return a clear error string.

Line 335, 825

address saleToken

```
function removeStuckTokens(address token, address
beneficiary, uint256 amount) external onlyAdmin {
```

saleToken can be directly cast to IERC20. It should be noted that the address is re-cast to address on a later line, which is redundant.

---

---

Line 403

```
require( sale.tokenPriceInAVAX.add(thirtyPercent) > price &&  
sale.tokenPriceInAVAX - thirtyPercent < price, "Price out of  
range." );
```

For consistency reasons, `safeMath` could be considered to be used throughout the whole contract, although we agree that for gas efficiency it could be absolved here as this cannot underflow. This is actually the case in all locations where `SafeMath` is not used therefore we will not insist on enforcing `SafeMath` in these locations but request you to go over the contract and verify the safety of these locations where `SafeMath` is not used as well.

Line 511

```
require(phaseId > uint256(Phases.Registration) && phaseId ==  
uint8(sale.phase), "Invalid phase.");
```

For consistency reasons, `Phases.Registration` should be casted with `uint8`.

Line 566, 725, 730

```
p.portionAmounts[i] += lastAmount;  
pBuyer.portionAmounts[portionId] += amountToSell;  
totalAmountExchanged += amountToSell;
```

`SafeMath` is not respected here. However, we believe this might not be an issue as the sum of the amounts should be capped to at most the sale amount.

Line 613

```
bool eligible;
```

The if statement below this line can be merged with this line to remove the if completely.

Line 710

```
_initParticipationForUser(buyer, 0, 0, block.timestamp,  
uint(sale.phase) /*==Phases.Idle*/);
```

We wonder whether it makes sense to increment `numberOfParticipants++` here, though this is likely misleading.

Several requirements lack a reversion string which might make it difficult to interpret a reversion reason within the explorers.

Many privileged functions are presently lacking events.

**Recommendation** Consider fixing the above issues.

**Resolution** PARTIALLY RESOLVED



---

## 2.2 SalesFactory

SalesFactory allows an administrator to deploy new sale contracts with a specific admin-definable implementation as a template using the clones pattern (non-upgradeable proxy contracts). These sale contracts are then initialized with certain variables and marked as having been created through this official factory. The newly created sale contract is then approved on the marketplace and added to the official list of all sale contracts. Only valid administrators are able to use this contract.





The list of privileged administrators are defined within the separate admin contract.

### 2.2.1 Privileged Functions

- `setAllocationStaking`
- `setAvalaunchMarketplace`
- `deploySale`
- `setImplementation`



## 2.2.2 Issues & Recommendations

<b>Issue #16</b>	<b>getAllSales has an incorrect input validation which causes the initial requirement to pass even though the endIndex is out of range</b>
<b>Severity</b>	 LOW SEVERITY
<b>Location</b>	<u>Line 124</u> <pre>require(endIndex &gt;= startIndex &amp;&amp; endIndex &lt;= allSales.length, "Invalid index range.");</pre>
<b>Description</b>	The input validation of getAllSales allows for the end index to be equal to the length of the allSales array. This end index does not exist and will therefore cause a later line of code to implicitly revert without any reversion message.
<b>Recommendation</b>	Consider adjusting the second portion of the requirement to be smaller than (<).
<b>Resolution</b>	 PARTIALLY RESOLVED
<b>Issue #17</b>	<b>Sales moderator is a fixed wallet which cannot be changed</b>
<b>Severity</b>	 LOW SEVERITY
<b>Description</b>	<p>SalesFactory defines a moderator which has privileges to manage individual sales. However, there is no way for the SalesFactory admins to change this moderator in case it is ever compromised or needs to be rotated.</p> <p>This is the case for the collateral variable as well — if the initialization ever needs to be done with a new collateral contract the whole factory needs to be redeployed.</p>
<b>Recommendation</b>	Consider either adding a setModerator and setCollateral function or setting the moderator to an administrative contract with a rotatable owner/multi-signature wallet.
<b>Resolution</b>	 RESOLVED A setModerator function has been introduced.

**Issue #18****Checks-effects-interactions pattern is not respected****Severity** LOW SEVERITY**Location**Line 90-103

```
(bool success, ) =  
sale.call( abi.encodeWithSignature( "initialize(address,addr  
ess,address,address,address)", address(admin),  
allocationStaking, collateral, address(marketplace),  
moderator ) );  
  
require(success, "Initialization failed."); // Mark sale as  
created through official factory  
  
isSaleCreatedThroughFactory[sale] = true; // Approve sale on  
marketplace marketplace.approveSale(sale);  
  
// Add sale to allSales  
  
allSales.push(sale);
```

**Description**

buyPortions does not adhere to the checks-effects-interactions standard.

**Recommendation**

Consider executing all external calls after the effects.

**Resolution** RESOLVED

CEI is now adhered to: isSalesCreatedThroughFactory and allSales.push calls have been moved above the initialization.

## Severity

 INFORMATIONAL

## Description

We have consolidated the typographical errors and the sections which can be further optimized for gas usage below.

Line 32

```
require(admin.isAdmin(msg.sender), "Only Admin can deploy sales");
```

This modifier has other uses apart from deploying sales. The error might therefore be incorrect in certain instances.

Line 37, 40 and 61

```
address _adminContract,  
address _marketplace,  
function setAvalaunchMarketplace(address _marketplace)  
external onlyAdmin {
```

These addresses can be cast as IAdmin and IAvalaunchMarketplace to avoid casting it later on. We do acknowledge that if it is provided as IAdmin, the non-zero check requires an additional cast which might explain the given approach.

Line 43, 44, 45, 56 and 62

```
require(_adminContract != address(0));  
require(_collateral != address(0));  
require(_moderator != address(0));  
require(_allocationStaking != address(0));  
require(_marketplace != address(0));
```

All of these requirements lack a reversion reason.

setAllocationStaking and setAvalaunchMarketplace lack events.

## Recommendation

Consider fixing the above typographical errors.

## Resolution

 PARTIALLY RESOLVED

Issue #20	Gas optimizations
Severity	<div>INFORMATIONAL</div>
Description	<p>We have consolidated the sections which can be further optimized for gas usage below.</p> <p><u>Line 10</u>  <code>IAdmin public admin;</code>    <code>admin</code> can be made immutable to save gas.</p> <p><u>Line 16</u>  <code>address public collateral;</code>    <code>collateral</code> can be made immutable to save gas.</p> <p><u>Line 18</u>  <code>moderator</code>    <code>moderator</code> can be made immutable to save gas, though it should likely be mutable.</p> <p><u>Line 116</u>  <code>if(allSales.length &gt; 0) return allSales[allSales.length - 1];</code>    <code>allSales.length</code> is fetched from storage twice within this function (in fact three times, as the return does a length check as well, but it is generally accepted to not over-optimize that in non-critical locations).</p>
Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	<div>PARTIALLY RESOLVED</div>



---

## 2.3 AvalaunchMarketplace

AvalaunchMarketplace is the marketplace contract where users offer their owned portions to buyers.

Each deployed AvalaunchSaleV2 contract will be approved on the marketplace during deployment. Users can then offer and remove their portions via `listPortions` and `removePortions`.


The `buyPortions` function allows users to purchase listed portions via a predefined configuration consisting of `owner`, `buyer`, `sale`, `portions`, `priceSum` and `sigExpTimestamp` which needs a valid off-chain signature by the admin of the marketplace. The desired portion will then get delisted from the marketplace and is being transferred from the old owner to the buyer via the `transferPortions` function in the corresponding AvalaunchSaleV2 contract.

Each purchase is to be made in AVAX and will be sent directly to the old owner after a fee is deducted. The fee can be freely set by the contract admin.

### 2.3.1 Privileged Functions

- `withdrawAVAX`
- `setFactory`
- `setFeeParams`
- `approveSale(factory | admin)`

## 2.3.2 Issues & Recommendations

<b>Issue #21</b>	<b>Signature lacks replay protection</b>
<b>Severity</b>	 <b>HIGH SEVERITY</b>
<b>Description</b>	<p>The only replay protection for the signature is sigExpTimestamp. This means that the provided transaction can be executed arbitrarily often until the timestamp is reached.</p> <p>Moreover, the signature lacks a chainId check. If the Avalaunch team ever decides to launch on another chain, the signature might also be valid there.</p> <p>Consider the following scenario:</p> <p>It is the year 2024, Avalaunch decided to launch their protocol on Ethereum Mainnet. Alice and Bob are participating in a new presale and Alice decides to list her portion. Bob wants to buy her portion for 100 ETH, the admin signs the transaction and Bob can execute it.</p> <p>Unfortunately, Alice also participated on the Avalanche chain and she listed the same portionId for a sale with the same address but she wants to sell it for 1000 AVAX. Bob can now execute the already signed transaction on Avalanche as well and purchase Alice's portion for 100 AVAX.</p> <p>This scenario has a relatively low likelihood, e.g. the sale contract must have the same address which means the factory must have the same address as well and it must be deployed with the same nonce.</p> <p>An issue with a higher likelihood is the fact that the signature can be replayed within the same contract as well: there is no "consumption" of signatures other than the expiration. This means that if a portion is sold from Alice to Bob, then Alice buys it back from Bob, Bob can potentially steal it from Alice at an unfavorable price by using the old signature.</p>
<b>Recommendation</b>	<p>Consider adding the chainId to the signature and mark the signature as used after the transaction was executed. Additionally, the provided timestamp should not be too loose.</p> <p>Consider using EIP-712 for signatures.</p>

---

**Resolution**

A check was added to ensure the signature can be used only once. The Avalaunch team also stated that a launch on another chain is highly unlikely hence the cross-chain replay is not a concern to them.

---

**Issue #22**

**Governance risk: Admin can frontrun any purchase with an increase of the fee parameters**

**Severity****Description**

Currently, the admin has the privilege to change the fee parameters without any limitation. They also have the privilege to reduce the price of any sale to zero.

While this itself is already an issue, the admin can frontrun any purchase, changing the fee parameters to a very high fee, resulting in a loss of funds for the portion owner.

**Recommendation**



Consider limiting the fee parameters to a reasonable value, consider managing the admin keys as described in the previous governance risk issue.



---


**Resolution**

The fee parameter is now limited to 5%.

---

Issue #23	There is no way to remove sale contracts
Severity	 LOW SEVERITY
Description	Currently, there is no possibility to remove sale contracts, once approved. Due to flexibility reasons, it might make sense to have a function to remove approved sale contracts.
Recommendation	Consider implementing a function that allows for removing approved sale contracts (if desired, as this might introduce stuck portions as a downside).
Resolution	 RESOLVED

Issue #24	Checks-effects-interactions pattern is not adhered to
Severity	 LOW SEVERITY
Description	<p>buyPortions does not adhere to the checks-effects-interactions standard:</p> <pre> IAvalaunchSaleV2(sale).transferPortions(owner, msg.sender, portions); // Compute fee amount uint256 feeAmount = msg.value.mul(feePercentage).div(feePrecision); // Increase total fees collected totalFeesCollected += feeAmount; </pre>
Recommendation	Consider executing the external call after the effects.
Resolution	 RESOLVED

**Issue #25****Contract deployment does not disable the initializer****Severity** LOW SEVERITY**Description**

There is no constructor code that disables the initializer. This results in the possibility of users initializing the implementation itself, which, in combination with payable functions, might lead to stuck or stolen AVAX if a user accidentally interacts with the initialized implementation directly.

**Recommendation**

Consider disabling the initializer during the contract deployment.

**Resolution** RESOLVED

An initializer blocking constructor has been added.



**Description**

We have consolidated the typographical errors into a single issue to keep the report brief and readable.

Line 34-36

```
event PortionListed(address portionOwner, address  
saleAddress, uint256 portionId);  
event PortionRemoved(address portionOwner, address  
saleAddress, uint256 portionId);  
event PortionSold(address portionSeller, address  
portionBuyer, address saleAddress, uint256 portionId);
```

It might be valuable to index the addresses of these events.

Line 51

```
function initialize(address _admin, address _factory,  
uint256 _feePercentage, uint256 _feePrecision) external  
initializer {
```

The first two addresses can be immediately cast to their desired types.

Line 163

```
emit SaleApproved(sale, block.timestamp);
```

Emitting a timestamp in an event seems rather redundant as this information is available in the log.

Line 179

```
function setFactory(address _factory) external onlyAdmin {
```

This variable can be directly provided as ISalesFactory.

---

withdrawAVAX, setFactory and setFeeParams lack events.

Some requirements lack reversion reasons which may be confusing to users.

---

**Recommendation** Consider fixing the typographical errors.

---

**Resolution**  RESOLVED





**PALADIN**  
BLOCKCHAIN SECURITY