PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Avalaunch

25 February 2022

paladinsec.co   info@paladinsec.co

Table of Contents

Paladin Blockchain Security

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

1.2   Contracts Assessed

Name	Contract	Live Code Match
Admin	Admin.sol	
Airdrop	Airdrop.sol	
AirdropAvax	AirdropAvax.sol	
AirdropSale	AirdropSale.sol	
AvalaunchSale	AvalaunchSale.sol	
SalesFactory	SalesFactory.sol	
XavaToken	0xd1c3f94DE7e5B45fa4eDBBA472491a9f4B166FC4	
DevToken	DevToken.sol	
AllocationStaking	AllocationStaking.sol	
AvalaunchBadgeFactory	AvalaunchBadgeFactory.sol	
Source	https://github.com/avalaunch-app/xava-protocol/tree/e43d174ecb64f00723516afc77a87b49198016c8	
Resolution	https://github.com/avalaunch-app/xava-protocol/tree/65b3779be43df907b1a14ce046dd990b68d19550	

1.3   Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	2	1	-	1
🟠 Medium	6	3	-	3
🟡 Low	10	4	-	6
🟣 Informational	45	12	4	29
Total	**63**	**20**	**4**	**39**

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.

Paladin Blockchain Security

1.3.1 Admin

ID	Severity	Summary	Status
01	INFO	`removeAdmin` reverts early when trying to remove admins due to an out-of-range exception	ACKNOWLEDGED
02	INFO	Excessive privilege: Any admin can remove and add admins	ACKNOWLEDGED
03	INFO	Lack of events for `addAdmin` and `removeAdmin`	ACKNOWLEDGED
04	INFO	Unnecessary and verbose usage of expensive while loops	ACKNOWLEDGED

1.3.2 Airdrop and AirdropAvax

ID	Severity	Summary	Status
05	INFO	SafeMath can be removed	ACKNOWLEDGED
06	INFO	`admin` and `airdropToken` can be made immutable	ACKNOWLEDGED
07	INFO	`safeTransfer` should be used within `withdrawTokens`	ACKNOWLEDGED
08	INFO	withdrawTokens can be made external	ACKNOWLEDGED
09	INFO	Usage of encodePacked is discouraged in critical code sections	ACKNOWLEDGED

1.3.3 AirdropSale

ID	Severity	Summary	Status
10	HIGH	Signature only validates the first amount, allowing an exploiter to withdraw all other ERC20 tokens in the contract freely	✔ RESOLVED
11	INFO	SafeMath can only be used for `uint256`	✔ RESOLVED
12	INFO	Certain variables can be made immutable	PARTIAL
13	INFO	Inconsistent usage of `beneficiary`	✔ RESOLVED

1.3.4 AvalaunchSale

ID	Severity	Summary	Status
14	MEDIUM	Contract does not inherit OpenZeppelin's upgradeable contract alternatives	RESOLVED
15	LOW	salesowner can mark tokens as deposited without transferring in any tokens if setSaleTokens was called before setSaleParams	RESOLVED
16	LOW	Lack of SafeMath usage	RESOLVED
17	LOW	Lack of on-chain overdraft protection	RESOLVED
18	LOW	PostponeSale can shift round.startTime beyond sale.saleEnd	RESOLVED
19	LOW	Governance can remove the sale.token	ACKNOWLEDGED
20	INFO	memory is used instead of calldata	PARTIAL
21	INFO	Certain functions have undetermined gas usage which could cause functions to become impossible to call	ACKNOWLEDGED
22	INFO	DepositTokens can unecessarily be called more than once	RESOLVED
23	INFO	_unlockingTimes should only be possible after sale.saleEnd	RESOLVED
24	INFO	Lack of events for certain functions	ACKNOWLEDGED
25	INFO	depositTokens does not work with fee on transfer tokens	ACKNOWLEDGED
26	INFO	Certain configurational functions remain callable even after the gate is closed	ACKNOWLEDGED
27	INFO	closeGate should only be callable once	RESOLVED
28	INFO	updateTokenPriceInAVAX will revert if _updateTokenPriceInAVAXPercentageThreshold is > 100	RESOLVED
29	INFO	withdrawUnusedFunds should only be called after the sale has ended	ACKNOWLEDGED
30	INFO	Usage of revert instead of require	RESOLVED
31	INFO	encodePacked should be avoided for signatures	RESOLVED

1.3.5   SalesFactory

ID	Severity	Summary	Status
32	LOW	Contract does not inherit all functions from ISalesFactory	ACKNOWLEDGED
33	INFO	Low-level clone logic is included directly in the contract	ACKNOWLEDGED
34	INFO	Unused variables: `saleOwnerToSale` and `tokenToSale` Unused event: `SaleOwnerAndTokenSetInFactory`	✔ RESOLVED
35	INFO	Lack of event for `setAllocationStaking`	PARTIAL
36	INFO	`admin` and `allocationStaking` can be made `immutable`	ACKNOWLEDGED
37	INFO	`setAllocationStaking` can be made `external`	✔ RESOLVED

1.3.6   XavaToken

ID	Severity	Summary	Status
38	INFO	`_decimals` can be made `immutable`	ACKNOWLEDGED
39	INFO	Several functions can be made `external`, and since the contract is deployed directly, the `virtual` keyword can be removed from all functions	ACKNOWLEDGED
40	INFO	Unused function: `_setupDecimals`	ACKNOWLEDGED
41	INFO	The contract does not contain `increaseAllowance` while it does contain `decreaseAllowance`	ACKNOWLEDGED
42	INFO	Gas optimization: Contract uses hardcoded strings in SafeMath functions	ACKNOWLEDGED

1.3.7   DevToken

Same as XavaToken above.

1.3.8 AllocationStaking

ID	Severity	Summary	Status
43	HIGH	Governance privilege: The contract is upgradeable which allows governance to withdraw all staked tokens	ACKNOWLEDGED
44	MEDIUM	LP tokens might not necessarily be equal to the reward token, which causes the contract to severely malfunction	ACKNOWLEDGED
45	MEDIUM	`emergencyWithdraw`, `deposit` and `withdraw` are prone to reentrancy attack	ACKNOWLEDGED
46	MEDIUM	The `deposit` and `fund` functions do not support fee on transfer tokens	✅ RESOLVED
47	MEDIUM	`verifySignature` never verifies the function name	ACKNOWLEDGED
48	LOW	Fees are still granted on the own share	ACKNOWLEDGED
49	LOW	The fee of the first deposit does not get added towards the `erc20Reward`	ACKNOWLEDGED
50	LOW	burnFromUser does not trigger within a deposit if `withdrawalFeePending` is greater than zero and `withdrawalFeeDepositAmount` is zero	ACKNOWLEDGED
51	LOW	The pending function will revert if totalAllocPoint is zero	ACKNOWLEDGED
52	INFO	Usage of `encodePacked` is discouraged in critical code sections	ACKNOWLEDGED
53	INFO	`safeTransfer` should be used within the `erc20Transfer` function	ACKNOWLEDGED
54	INFO	SafeMath is not used	ACKNOWLEDGED
55	INFO	Certain functions can be made external	ACKNOWLEDGED
56	INFO	`salesRegistered` can only be viewed on the contract in the `userInfo` struct, however it is not possible to view it on the front-end	ACKNOWLEDGED
57	INFO	Lack of events for certain functions	ACKNOWLEDGED
58	INFO	Lack of validation: `startTimestamp` should be in the future; `add` function has no check for existing tokens	ACKNOWLEDGED
59	INFO	Unnecessary use of `address(msg.sender)`	ACKNOWLEDGED

1.3.9 AvalaunchBadgeFactory

ID	Severity	Summary	Status
60	MEDIUM	`mintBadges` mint receipt hook has an outdated `badgeIdMintedSupply` which can be a cause of exploits in derivative contracts	✔ RESOLVED
61	INFO	Certain functions can be made external	✔ RESOLVED
62	INFO	Lack of events for certain functions	PARTIAL
63	INFO	Gas optimization: Usage of `uint32` has causes extra gas usage	✔ RESOLVED

2   Findings

2.1   Admin

The admin contract is a dependency used to define and remove admins as well as view all current admins. The deployer can add admins during the creation of the contract.

2.1.1   Privileged Roles

The following functions can be called by the owner:

- `addAdmin`

- `removeAdmin`

2.1.2   Issues & Recommendations

Issue #01	removeAdmin reverts early when trying to remove admins due to an out-of-range exception
Severity	● INFORMATIONAL

Location

<u>Lines 51-58</u>

```
uint i = 0;

while(admins[i] != _adminAddress) {
    if(i == admins.length) {
        revert("Passed admin address does not exist");
    }
    i++;
}
```

Description

The removeAdmin function loops over the admins to find the admin index to remove (the location in the list of admins). However, this looping behavior is flawed in case the admin does not exist in this list. In this case, admins[i] would go out of range. The lines of code that revert with "Passed admin address does not exist" can therefore never be reached. It should also be noted that these lines of code can never be reached anyways, since the functions starts with an isAdmin requirement.

Recommendation

Consider either using EnumerableSet to remove this while loop. Alternatively, the if statement can be removed completely because the isAdmin requirement already ensures that the array must contain the admin at this point.

Resolution

● ACKNOWLEDGED

Issue #02	Excessive privilege: Any admin can remove and add admins
Severity	INFORMATIONAL
Description	Presently, any admin can add other admins and even remove existing ones. If one of the admins ever turns malicious, they could therefore remove everyone else and remain as the only admin in the system. No admins can be added by the honest parties at this point, nor can they remove the malicious admin.
Recommendation	Consider having an `owner` role, which will be the only role that can add or remove admins (this role can alternatively be called `ADMIN_MANAGEMENT`). It should be noted that such role-based management is easier done using OpenZeppelins RBAC solutions.
Resolution	ACKNOWLEDGED

Issue #03	Lack of events for `addAdmin` and `removeAdmin`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions. Consider also adding the `AdminAdded` event to the constructor when admins are added.
Resolution	ACKNOWLEDGED

Issue #04	Unnecessary and verbose usage of expensive while loops
Severity	INFORMATIONAL
Description	The `while` loop can cost a lot of gas while executing. Within the `remove` function, as shown in a previous issue, it is used to find the index of an admin in the internal array (list) of admins. However, such logic is unnecessary as there exist common libraries by OpenZeppelin that abstract away such logic and furthermore do not use looping for removal. Instead, these libraries allow removal of elements in O(1). It should also be noted that `getAllAdmins` can run out of gas as well (or the RPC doesn't allow returning it), once the admin array becomes too large.
Recommendation	Consider using the EnumerableSet library by OpenZeppelin. Alternatively and perhaps even more ideally, one can consider using the RBAC solutions by OpenZeppelin, which render this whole contract redundant. Consider adding pagination to `getAllAdmins`.
Resolution	ACKNOWLEDGED

2.2   Airdrop and AirdropAvax

The Airdrop contract allows users to claim tokens for which they are eligible. Governance can set the token to be airdropped during contract creation and need to provide signatures of the airdrop allocations off-chain, which will be validated on-chain when a user claims their airdrop. A signature by any of the registered admins in the admin contract is valid. Each user can only claim once and signatures are specific to the user.

This audit section has been combined with the AirdropAvax contract section in an effort to manage the audit report size and to keep it accessible for all readers. Both contracts are extremely similar and no additional issues were found within AirdropAvax. Compared to Airdrop, AirdropAvax grants airdropped Avax tokens.

It should be noted that the zero address must never be added as an admin, as is currently forbidden in the Admin contract that was audited by Paladin. This is because any wrong signature will be marked as signed by this address.

2.2.1   Issues & Recommendations

Issue #05	SafeMath can be removed
Severity	INFORMATIONAL
Location	Line 11 `using SafeMath for *;`
Description	Right now SafeMath is used for every variable type within the contract (*). This makes little sense as SafeMath is only designed to work for `uint256`. In addition, SafeMath is not used anywhere within this contract.
Recommendation	Consider removing SafeMath.
Resolution	ACKNOWLEDGED

Issue #06	`admin` and `airdropToken` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	ACKNOWLEDGED

Airdrop and AirdropAvax   Paladin Blockchain Security

Issue #07	safeTransfer should be used within withdrawTokens
Severity	● INFORMATIONAL

Location

<u>Lines 31-45</u>

```solidity
function withdrawTokens(bytes memory signature, uint256
amount) public {
    require(msg.sender == tx.origin, "Require that message
sender is tx-origin.");

    address beneficiary = msg.sender;

    require(checkSignature(signature, beneficiary, amount),
"Not eligible to claim tokens!");
    require(!wasClaimed[beneficiary], "Already claimed!");
    wasClaimed[msg.sender] = true;

    bool status = airdropToken.transfer(beneficiary,
amount);
    require(status, "Token transfer status is false.");

    totalTokensWithdrawn = totalTokensWithdrawn.add(amount);
    emit TokensAirdropped(beneficiary, amount);
}
```

Description

In the withdrawTokens function, the transfer method is used to transfer tokens. This will not work for non-compliant tokens without a return value.

Recommendation

Consider using safeTransfer instead of transfer.

Resolution

● ACKNOWLEDGED

Issue #08	withdrawTokens can be made external

Severity	🟣 INFORMATIONAL

Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.

Recommendation	Consider marking the variable as external. The contract can then be optimized for gas by replacing all memory sections with calldata (this might require some redesign).

Resolution	⚫ ACKNOWLEDGED

Issue #09	Usage of encodePacked is discouraged in critical code sections

Severity	🟣 INFORMATIONAL

Location	`Line 49` `bytes32 hash = keccak256(abi.encodePacked(beneficiary, amount, address(this)));`

Description	The signature validation scheme checks the signature over a collection of bytes which is tightly packed. This is however not encouraged for critical sections of code as it could allow for hash collisions. This issue has been marked as informational as hash collisions are mainly an issue with variable length values (strings...) and the above code section does not have these. We therefore do not believe that there is any way to abuse this hash but would still like to recommend the best practice which more effectively guarantees this.

Recommendation	Consider using abi.encode instead of abi.encodePacked.

Resolution	⚫ ACKNOWLEDGED

2.3   AirdropSale

The AirdropSale contract allows to airdrop multiple tokens as well as the native gas token (AVAX) within a single airdrop transaction. It is very similar to Airdrop and AirdropAvax in its structuring and implementation. We refer to those sections of the report for further information about the airdrop mechanism.

All issues from Airdrop and AirdropAvax equally apply to this contract. To keep this report brief and readable to third parties, they have not been repeated here.

2.3.1   Issues & Recommendations

Issue #10	Signature only validates the first amount, allowing an exploiter to withdraw all other ERC20 tokens in the contract freely
Severity	🔴 HIGH SEVERITY
Location	Line 63 `require(checkSignature(signature, beneficiary, amounts[0]), "Not eligible to claim tokens!");`
Description	The AirdropSale contract presently only validates that the user is actually eligible to receive the first token amount. However, as many tokens are distributed to users, the user can specify any and all amounts for the other tokens without the function reverting. A malicious user will simply add the total amount of tokens as their "airdrop allocation" and they will receive the total airdropped supply.
Recommendation	Consider updating the `checkSignature` scheme to firstly use encode instead of `encodePacked`, and secondly to validate the whole `amounts` array.
Resolution	✅ RESOLVED

Issue #11	SafeMath can only be used for `uint256`
Severity	⬤ INFORMATIONAL
Location	Line 12 `using SafeMath for *;`
Description	Right now SafeMath is used for every variable type within the contract (*). This makes little sense as SafeMath is only designed to work for `uint256`.
Recommendation	Consider using SafeMath only for `uint256` instead of * `using SafeMath for uint256;`
Resolution	✔ RESOLVED

Issue #12	Certain variables 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: ⁻ `admin` ⁻ `_includesAvax` ⁻ `includesERC20s` This is considered best practice since it makes the code more accessible for third party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	⬤ PARTIALLY RESOLVED

Issue #13	Inconsistent usage of beneficiary
Severity	● INFORMATIONAL
Location	Line 67 `wasClaimed[msg.sender] = true;`
Description	Within `withdrawTokens`, `msg.sender` gets assigned to the beneficiary variable, however it is not used throughout the whole function as it should be. Specifically, within `wasClaimed`, `msg.sender` is still used.
Recommendation	Consider using `beneficiary` throughout the function.
Resolution	✔ RESOLVED

2.4   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).

2.4.1 Privileged Roles

The following functions can be called by the owner:

- setVestingParams

- shiftVestingUnlockTimes

- setSaleParams

- setSaleToken

- setRounds

- updateTokenPriceInAvax

- postponeSale

- extendRegistrationPeriod

- setCapPerRound

- withdrawEarningsAndLeftover

- withdrawEarnings

- withdrawLeftover

- withdrawRegistrationFees

- withdrawUnusedFunds

2.4.2   Issues & Recommendations

Issue #14	Contract does not inherit OpenZeppelin's upgradeable contract alternatives
Severity	🔴 MEDIUM SEVERITY
Location	Line 13 `contract AvalaunchSale is Initializable, ReentrancyGuard {`
Description	Even though the contract is deployed as a proxy clone, it currently does not inherit from the upgradeable OpenZeppelin contracts. This causes the constructor of both dependencies to never be called. This issue is marked as Medium compared to High as within the present ReentrancyGuard implementation, the constructor does not strictly need to be called. This might not be the case for all implementations however.
Recommendation	Consider using the upgradeable dependencies.
Resolution	✅ RESOLVED ReentrancyGuard has been removed as a dependency.

Issue #15	salesowner can mark tokens as deposited without transferring in any tokens if setSaleTokens was called before setSaleParams
Severity	🟡 LOW SEVERITY
Description	The contract contains a function to override the sale token in emergencies. If this function is called before setSaleParams, this would allow the salesowner to deposit zero tokens but still mark the contract as deposited, which might mislead other system components.
Recommendation	Consider requiring the sales params to be set before the sale token can be set.
Resolution	✅ RESOLVED depositTokens() now validates that the sale parameters have been set.

Issue #16	Lack of SafeMath usage
Severity	🟡 LOW SEVERITY
Location	Line 193 sum += _percents[i]; Line 453 round.startTime + timeToShift < sale.saleEnd
Description	Using raw addition or subtraction methods instead of SafeMath can result in underflows or overflows. This issue is marked as low severity as the user cannot abuse these portions of code — they are limited to configurational sections of the contract.
Recommendation	Consider using SafeMath throughout the contract.
Resolution	✅ RESOLVED

AvalaunchSale Paladin Blockchain Security

Issue #17	Lack of on-chain overdraft protection
Severity	🟡 LOW SEVERITY
Description	Presently the contract lacks any notion of on-chain "sale cap". This logic is presumably handled off-chain.
Recommendation	Consider adding a basic requirement in participate that no more than the total sum of tokens can be handed out.
Resolution	✅ RESOLVED The following validation was added which causes participation to revert in case it causes excess allocation: `amountOfTokensBuying <= sale.amountOfTokensToSell.sub(sale.totalTokensSold)`

Issue #18	PostponeSale can shift round.startTime beyond sale.saleEnd
Severity	🟡 LOW SEVERITY

Location	<u>Lines 451–455</u>
	`round.startTime = round.startTime.add(timeToShift);`
	`require(`
	`    round.startTime + timeToShift < sale.saleEnd,`
	`    "Start time can not be greater than end time."`
	`);`

Description	`round.startTime` gets extended by `timeToShift` before the requirement takes place. This causes the requirement to use an already increased `startTime` and revert in cases where it should not.
Recommendation	Consider placing the requirement before the extending of the variable. The requirement should also use SafeMath once these are inverted (although not strictly necessary since `.add` is still eventually called).
Resolution	✅ RESOLVED

Issue #19	Governance can remove the `sale.token`

Severity	🟡 LOW SEVERITY

Location	<u>Lines 274-282</u>

```
function setSaleToken(
    address saleToken
)
    external
    onlyAdmin
    onlyIfGateOpen
{
    sale.token = IERC20(saleToken);
}
```

Description

The function `removeStuckTokens` allows the governance to remove any token in the contract besides the `sale.token`. However, it is possible to change `sale.token` in the function `setSaleToken` to some other token if the gate is still open and therefore drain the actual `sale.token`.

This issue is marked as low severity given the reputation of the client. If parties are unsure about the key management or reputation of the client, they should of course still take this issue seriously.

Recommendation

Consider only allowing `setSaleToken` to be called if it is presently set to zero.

Resolution

⚫ ACKNOWLEDGED

Issue #20	memory is used instead of calldata

Severity	🟣 INFORMATIONAL

Location	Example: Line 363

```
function registerForSale(bytes memory signature, uint256
roundId)
```

Example: Line 513
```
function participate(
    bytes memory signature,
    uint256 amount,
    uint256 amountXavaToBurn,
    uint256 roundId
)
```

Description	Compared to the keyword memory, the use of calldata is considered as best practice and saves gas. This is possible because the EVM will directly access the bytes from the calldata instead of first loading them into memory. The advantage of memory comes into play if you need to actually edit certain portions of the memory, as calldata is of course immutable.

Recommendation	Consider using calldata instead of memory and rewriting the contract to use calldata throughout all functions with immutable bytes user inputs.

Resolution	🔵 PARTIALLY RESOLVED

calldata has been introduced in certain locations.

Issue #21	Certain functions have undetermined gas usage which could cause functions to become impossible to call
Severity	🟣 INFORMATIONAL
Location	Line 319 `function setRounds`
Description	The function `setRounds` allows governance to set `startTimes` and `maxParticipations` for each round. However, if in any case those arrays are very long, it is possible that the function runs out of gas. Since it is very unlikely that so many rounds are added, this will remain informational.
Recommendation	Consider adding a cap for the array lengths.
Resolution	⚫ ACKNOWLEDGED

Issue #22	DepositTokens can unecessarily be called more than once
Severity	🟣 INFORMATIONAL
Location	Line 501 `sale.tokensDeposited = true;`
Description	In the function, `sale.tokensDeposited` is set to `true` — we believe that the contract creator had the intention to make this function only callable once, however, there is no check if `sale.tokensDeposited is != true`.
Recommendation	Consider adding `require(!sale.tokensDeposited, "...")` at the beginning of the function.
Resolution	✅ RESOLVED The recommended requirement has been introduced.

Severity	● INFORMATIONAL
| Code | ```
function setVestingParams(
    uint256[] memory _unlockingTimes,
    uint256[] memory _percents,
    uint256 _maxVestingTimeShift
)
    external
    onlyAdmin
{
    require(
        vestingPercentPerPortion.length == 0 &&
        vestingPortionsUnlockTime.length == 0
    );
    require(_unlockingTimes.length == _percents.length);
    require(portionVestingPrecision > 0, "Safeguard for
making sure setSaleParams get first called.");
    require(_maxVestingTimeShift <= 30 days, "Maximal shift
is 30 days.");

    // Set max vesting time shift
    maxVestingTimeShift = _maxVestingTimeShift;

    uint256 sum;

    // Set vesting portions percents and unlock times
    for (uint256 i = 0; i < _unlockingTimes.length; i++) {
        vestingPortionsUnlockTime.push(_unlockingTimes[i]);
        vestingPercentPerPortion.push(_percents[i]);
        sum += _percents[i];
    }

    require(sum == portionVestingPrecision, "Percent
distribution issue.");
}
``` |
| Description | Each sale gets its specific _unlockTimes[] as an array. However, all unlockTimes should logically only occur after sale.saleEnds. |
| Recommendation | Consider adding a require statement to validate this: `require(_unlockTime[0] > sale.SaleEnds)` |

Resolution	✅ RESOLVED
	The recommended check alongside another safety check that guarantees that the unlock times are in increasing order have been added.

Issue #24	Lack of events for certain functions
Severity	🟣 INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications:

- `setVestingParams`
- `shiftVestingUnlockingTimes`
- `setSaleToken`
- `postponeSale`
- `extendRegistrationPeriod`
- `depositTokens`
- `withdrawEarningsInternal`
- `withdrawLeftoverInternal`
- `withdrawRegistrationFees`
- `removeStuckTokens`
- `withdrawUnusedFunds`
- `setUpdateTokenPriceInAVAXParams`

Recommendation	Add events for the above functions.
Resolution	⚫ ACKNOWLEDGED

Issue #25	depositTokens does not work with fee on transfer tokens
Severity	● INFORMATIONAL

Location	<u>Lines 504-508</u>

```
sale.token.safeTransferFrom(
    msg.sender,
    address(this),
    sale.amountOfTokensToSell
);
```

Description	During `safeTransferFrom`, the `sales.amountOfTokensToSell` is transferred from the `msg.sender` to the contract. However, with a fee on transfer token, the contract would not receive the desired amount.
Recommendation	Consider avoiding fee on transfer tokens or exclude the contract from the transfer tax if a token like that is ever used.
Resolution	● ACKNOWLEDGED

Issue #26	Certain configurational functions remain callable even after the gate is closed
Severity	INFORMATIONAL
Description	Presently functions like `shiftVestingUnlockingTime` and `extendRegistrationPeriod` can still be called after the gate is closed. Closing the gate is supposed to lock in most of the configurational aspects, hence we believe functions like the above two might have accidentally been excluded from an `isGateClosed` modifier.
Recommendation	Consider whether these functions should be called after the gate is closed. If not, add a modifier. The client should also go over the whole contract as this list is non-exhaustive, the two functions above were just the most likely to not be needed after the gate is closed.
Resolution	ACKNOWLEDGED

Issue #27	closeGate should only be callable once

Severity

● INFORMATIONAL

Location

Lines 906-927

```
function closeGate() external onlyAdmin {
    // Require that sale is created
    require(sale.isCreated, "closeGate: Sale not created.");
    // Require that sale token is set
    require(address(sale.token) != address(0), "closeGate:
Token not set.");
    // Require that tokens were deposited
    require(sale.tokensDeposited, "closeGate: Tokens not
deposited.");
    // Require that token price updating params are set
    require(
        updateTokenPriceInAVAXPercentageThreshold != 0 &&
updateTokenPriceInAVAXTimeLimit != 0,
        "closeGate: Params for updateTokenPriceInAvax not
set."
    );
    // Require that registration times are set
    require(
        registration.registrationTimeStarts != 0 &&
registration.registrationTimeEnds != 0,
        "closeGate: Registration params not set."
    );

    // Close the gate
    gateClosed = true;
    emit GateClosed(block.timestamp);
}
```

Description

Currently, it is possible to call the function `closeGate` more than once. From a logical point of view, It does not make any sense to call this function more than once.

Recommendation

Consider adding `require(!gateClosed)` at the beginning of the function.

Resolution

✔ RESOLVED

The function can now only be called when the gate is open.

Issue #28	**updateTokenPriceInAVAX will revert if _updateTokenPriceInAVAXPercentageThreshold is > 100**
Severity	🟣 INFORMATIONAL
Location	Line 422 `price > sale.tokenPriceInAVAX.sub(maxPriceChange),`
Description	The function setUpdateTokenPriceInAVAXParams allows the variable updateTokenPriceInAVAXPercentageThreshold to be > 100. If that is the case, it is not possible to execute the function updateTokenPriceInAVAX since it reverts in the line mentioned above.
Recommendation	Consider adding `require(updateTokenPriceInAVAXPercentageThreshold <= 100)` at the beginning of the setUpdateTokenPriceInAVAXParams function.
Resolution	✅ RESOLVED The recommended requirement has been added to setUpdateTokenPriceInAVAXParams.

Issue #29	**withdrawUnusedFunds should only be called after the sale has ended**
Severity	🟣 INFORMATIONAL
Description	As the function name indicates, this function is intended for the withdrawal of funds which are not used, and should therefore only be callable after the sale has ended.
Recommendation	Consider adding `require(block.timestamp >= sale.saleEnd)` at the beginning of the function.
Resolution	⚫ ACKNOWLEDGED

Issue #30	Usage of `revert` instead of `require`
Severity	● INFORMATIONAL

Location	

```
Line 641
{
    p.isPortionWithdrawn[portionId] = true;
    uint256 amountWithdrawing = p
        .amountBought
        .mul(vestingPercentPerPortion[portionId])
        .div(portionVestingPrecision);

    // Withdraw percent which is unlocked at that portion
    if(amountWithdrawing > 0) {
        sale.token.safeTransfer(msg.sender,
amountWithdrawing);
        emit TokensWithdrawn(msg.sender, amountWithdrawing);
    }
} else {
    revert("Tokens already withdrawn or portion not unlocked
yet.");
}
```

Description	It is not necessary to extend the code through a `revert` pattern; instead, a `require` pattern is the better and cleaner pratice.

Recommendation	Consider using a `require` pattern instead of `revert`.

Resolution	✔ RESOLVED
	The client now follows best practice by using a requirement instead of the `if` statement.

Issue #31	encodePacked should be avoided for signatures
Severity	● INFORMATIONAL
Description	The encodePacked function concatenates all data into one long string. It is known to cause hash duplicates for variable length concatenations. For example ab + c will have the same hash as a + bc. Within the encode function, variables are more nicely separated and hash collisions are generally avoided. This issue has been marked as informational as the signatures in question do not contain collisions as far as we are aware. However, as encode is easier to justify and generally more trusted in these scenarios, it is still recommended.
Recommendation	Consider using encode instead of encodePacked.
Resolution	✔ RESOLVED The client has indicated that they have carefully evaluated the encoding of encodePacked and that it does not pose an issue here. Paladin has undergone thorough testing of this functionality as well and believes the usage of encodePacked is all right, although we still prefer our clients to be safe and go with encode whenever they can.

2.5   SalesFactory

The SalesFactory contract allows governance to deploy new AvalaunchSale contracts. These contracts are used by individual project token sales on avalaunch. For a further description of such sales we refer to the AvalaunchSale section. This contract finally also keeps a record of all sales in existence.

2.5.1   Privileged Roles

The following functions can be called by the owner:

- `setAllocationStaking`

- `deploySale`

- `setImplementation`

2.5.2   Issues & Recommendations

Issue #32	Contract does not inherit all functions from ISalesFactory
Severity	🟡 LOW SEVERITY
Description	The audit contracts contain an interface called ISalesFactory. We however believe that this interface is outdated as `setSaleOwnerAndToken` is not implemented within SalesFactory.
Recommendation	Consider explicitly implementing ISalesFactory and updating it to match the correct functions.
Resolution	⚫ ACKNOWLEDGED

Issue #33	Low-level clone logic is included directly in the contract
Severity	🟣 INFORMATIONAL
Description	The contract includes low-level clone logic to allow for proxy clones. Proxy clones are contracts that function like an upgradeable proxy but without the upgradeability. They are used to save on gas and refer to common implementations. Within the SalesFactory, the cloning logic is copied in directly, which can be verbose for third-party validators. Instead, by using a library like OpenZeppelin, most validators will immediately understand that this low-level code is correct.
Recommendation	Consider using OpenZeppelin Clones: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/proxy/Clones.sol Also consider rewriting the low level `.call` to a normal `initialize` call by wrapping the sale address in an interface.
Resolution	⚫ ACKNOWLEDGED

Issue #34	Unused variables: `saleOwnerToSale` and `tokenToSale` Unused event: `SaleOwnerAndTokenSetInFactory`
Severity	🟣 INFORMATIONAL
Description	Variables defined in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length and bytecode size unnecessarily.
Recommendation	Consider removing the aforementioned variables to keep the contract short and simple. The event `SaleOwnerAndTokenSetInFactory` can also be removed as this function appears to have been deleted and is no longer in use.
Resolution	✅ RESOLVED

Issue #35	Lack of event for `setAllocationStaking`
Severity	🟣 INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	🔵 PARTIALLY RESOLVED An event was added and is now emitted in the constructor. However, it is not emitted in `setAllocationStaking`.

Issue #36	admin and allocationStaking can be made immutable
Severity	● INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making the above variables explicitly immutable.
Resolution	● ACKNOWLEDGED

Issue #37	setAllocationStaking can be made external
Severity	● INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as external.
Resolution	✔ RESOLVED

2.6   XavaToken

The Xava Token is a simple ERC20 token with no minting function present. However, it extends the standard ERC20 token by allowing anyone to call burn, which removes the tokens out of circulation. This can only be done for the user's own balance.

The deployer can choose the total supply during contract creation. This supply is minted to the deployer.

2.6.1   Token Overview

Address	0xd1c3f94DE7e5B45fa4eDBBA472491a9f4B166FC4
Token Supply	100,000,000
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None
Pre-mints	100,000,000

2.6.2   Issues & Recommendations

Issue #38	`_decimals` 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	● ACKNOWLEDGED

Issue #39	**Several functions can be made external, and since the contract is deployed directly, the `virtual` keyword can be removed from all functions**
Severity	🟣 INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword: ⁻ `transfer` ⁻ `approve` ⁻ `burn` ⁻ `transferFrom` ⁻ `decreaseAllowance` ⁻ `name` ⁻ `allowance` ⁻ `symbol` ⁻ `decimals` ⁻ `totalSupply` ⁻ `balanceOf` Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the above variables as `external` and consider removing the `virtual` keyword from all functions.
Resolution	⚫ ACKNOWLEDGED

Issue #40	Unused function: `_setupDecimals`
Severity	INFORMATIONAL
Description	Functions defined in a contract but not used within the contract could confuse third-party auditors. They also increase the contract length and bytecode size unnecessarily.
Recommendation	Consider removing the function to keep the contract short and simple.
Resolution	ACKNOWLEDGED

Issue #41	The contract does not contain `increaseAllowance` while it does contain `decreaseAllowance`
Severity	INFORMATIONAL
Description	Front-running became a very common problem when changing the allowance which can lead to the problem of double spending. Therefore, the functions `increaseAllowance` and `decreaseAllowance` exist. From a logical and security point of view the contract should not only contain `decreaseAllowance` but also `increaseAllowance`.
Recommendation	Consider adding a `increaseAllowance` function similar to `decreaseAllowance`.
Resolution	ACKNOWLEDGED

Issue #42	Gas optimization: Contract uses hardcoded strings in SafeMath functions
Severity	● INFORMATIONAL
Location	XavaToken::45 (Example) _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
Description	The contract injects the error message into SafeMath. This is known to cost extra gas, even on the happy path, as it causes memory allocation.
Recommendation	Consider checking the identity explicitly using a `require` statement and then using non-SafeMath to do the subtractions and additions instead. SafeMath has also created the `trySub` and `tryAdd` functions in more recent versions to address this gas usage concern.
Resolution	● ACKNOWLEDGED

2.7   DevToken

The DevToken is a token which was included within the audit scope with no special features. We expect it to be used for testing. As the source code is identical to XavaToken, we refer to the XavaToken section of this report for the list of issues and privileges that apply.

2.8   AllocationStaking

AllocationStaking is a Masterchef-like contract where users can stake `lpToken` to earn a token to be set (erc20). Contrary to the traditional Masterchef they can extend the reward time through a 'fund' function. The lpTokens will furthermore get locked for a timeframe to be set in the 'setTokensUnlockTime' function, another thing which should not be forgotten to mention is the distribution of the deposit fee amongst the stakers in the 'updatePoolWithFee' function. A deposit and withdrawal fee is only settable for the first pool, they can each be set up to 100% however.

It should furthermore be noted that withdrawals require an off-chain governance signature. Emergency withdrawals are still allowed without such a signature.

Finally, the withdrawal fee on the zero pool reduces linearly over a configurable time-period.

2.8.1   Privileged Roles

The following functions can be called by the owner:

- `setSalesFactory`

- `add`

- `setDepositFee`

- `set`

- `setPostSaleWithdrawPenaltyPercentAndLength`

- `setTokensUnlockTime (factory)`

- `redistributeXava (factory)`

2.8.2   Issues & Recommendations

Issue #43	Governance privilege: The contract is upgradeable which allows governance to withdraw all staked tokens
Severity	🔴 HIGH SEVERITY
Description	The contract can be upgraded at any time, therefore malicious functions might be added which can result in a total loss of user funds. This could happen by less reputable projects in an attempt to exit with a profit when the project doesn't go as well as expected. However, given that Avalaunch is quite reputable this is less of a risk.
	This issue is still marked as high risk since the possibility of keys getting stolen or ending up in the wrong hands remains present. Other governance privileges in the contract include:
	- Deposit fees can be set up to 100%
	- Tokens could become locked forever
	- Withdrawal fees can be set up to 100%
	- Lack of cap on the withdrawal penalty length (the duration)
Recommendation	Consider whether upgradeability is desired, if so, consider setting the admin to a multi-sig with doxxed participants.
Resolution	⚫ ACKNOWLEDGED
	The client has indicated that they acknowledge this issue and are in the process of setting up a new governance structure where they can give specific wallets and contracts specific rights. Their goal is to minimize the governance risk but as the contracts will remain upgradable, larger investors should carefully assess the current proxy admin who can upgrade these contracts.

Issue #44	LP tokens might not necessarily be equal to the reward token, which causes the contract to severely malfunction
Severity	🔴 MEDIUM SEVERITY
Description	Within the XAVA distribution and compound functions, it is assumed that all pools have the native token as their LP token, but this is not guaranteed. In case this is not the case, the contract will compound the native token into non-native pools.
Recommendation	Consider removing the `lpToken` variable and always using `erc20` (a slight misnomer, `rewardToken` would be more adequate) as the `lpToken`.
Resolution	⚫ ACKNOWLEDGED
	The client has already carefully validated that these tokens are the same within their process and would rather not make changes to code which works within this process.

Issue #45	**emergencyWithdraw, deposit and withdraw are prone to reentrancy attack**
Severity	🔴 MEDIUM SEVERITY
Location	Lines 395-407
Description	The emergencyWithdraw function is used to allow the user to withdraw funds without claiming rewards. However, it is vulnerable to reentrancy attacks because the amount is only reset after the token has been transferred. If a token allows for external code execution, emergencyWithdraw could be called twice to withdraw the user.amount twice.
	To reiterate, in the emergencyWithdraw function, the lpTokens are being transferred before the user.amount is set to zero. During a reentrancy attack, a malicious hacker can use this vulnerability to withdraw more tokens than he actually owns and therefore drain the whole pool.
	However, this is only possible with tokens that are vulnerable to reentrancy and since this contract is designed to use the XAVA Token (which is not vulnerable to this) as the main staking token we will only mark this issue as medium severity.
	Furthermore, within deposit and withdraw, the rewardDebt is only updated after external calls have been made. If reentrancy is permitted on these external calls (unlikely), then the rewards could be inflated.
Recommendation	Consider adding a reentrancy modifier and changing the logic of the function by first setting the user.amount to zero and then only transfer the lpTokens afterwards.
	Consider adding reentrancy guards to deposit and withdraw or rewriting them to adhere to checks-effects-interactions.
Resolution	⚫ ACKNOWLEDGED
	Avalaunch has indicated that they carefully vet the tokens they add to their staking contract and have no intention to add more complex tokens which could introduce reentrancy risk.
	Given careful vetting of tokens, this issue will therefore not present itself as it requires a token which allows for reentrancy.

Issue #46	**The deposit and fund functions do not support fee on transfer tokens**
Severity	🟠 MEDIUM SEVERITY

Location	Lines 395-407

```
function deposit(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    [...]

    if (user.amount > 0) {
        uint256 pendingAmount =
user.amount.mul(pool.accERC20PerShare).div(1e36).sub(user.re
wardDebt);
        erc20Transfer(msg.sender, pendingAmount);
    }
    pool.lpToken.safeTransferFrom(address(msg.sender),
address(this), _amount);
    pool.totalDeposits =
pool.totalDeposits.add(depositAmount);

    user.amount = user.amount.add(depositAmount);
    user.rewardDebt =
user.amount.mul(pool.accERC20PerShare).div(1e36);
    emit Deposit(msg.sender, _pid, depositAmount);
}
```

Description	Fee on transfer tokens differ from normal tokens in that the amount sent does not equal the amount received. This often causes pools to get exploited and drained due to a miscalculation of user.amount.

If the pool is using a fee on transfer token as lpToken, the contract will only receive the amount after the transfer tax is deducted during a deposit but the user.amount variable does not reflect this loss. Due to this, there will not be enough tokens within the pool to repay everyone and withdrawals might eventually break.

In addition, the fund function does not support fee on transfer tokens either.

AllocationStaking Paladin Blockchain Security

Recommendation	Consider adding a before-after check:

```
uint256 balanceBefore =
pool.lpToken.balanceOf(address(this));

pool.lpToken.safeTransferFrom(msg.sender, address(this),
_amount);

depositAmount = pool.lpToken.balanceOf(address(this)) -
balanceBefore;
```

Resolution	✔️ RESOLVED
	The client has indicated that they will not support such tokens. Given that the impact is much more modest than the previously acknowledged issues, we are marking this issue as resolved.

Issue #47	**verifySignature never verifies the function name**
Severity	🔴 MEDIUM SEVERITY
Description	The verifySignature method does not verify the provided function name. If it was ever used for multiple functions, this could allow for replay attacks.
Recommendation	Consider adding the function name hash to the signature hash.
Resolution	⚫ ACKNOWLEDGED
	As this issue only really presents itself if the contract is extended, the client has decided to not fix it yet.

Issue #48	Fees are still granted on the own share

Severity

🟡 LOW SEVERITY

Location

Lines 426-452

```
function compound(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    [...]

    // Update accounting around burns
        burnFromUser(msg.sender, _pid, fee);
    // Update pool including fee for people currently staking
        updatePoolWithFee(_pid, fee);

    [...]

    emit CompoundedEarnings(msg.sender, _pid,
amountCompounding, user.amount);
    }
```

Description

After the user's deposit fee gets deducted, the contract calls the
updatePoolWithFee function. This results in the increment of
pool.accERC20PerShare, therefore the user will receive a portion of
his deposit fee back.

The issue with the current design is that within the
redistributeXava function, the amount to redistribute is still
included in the user stake. They therefore receive back a larger part
of their fees than they should.

Recommendation

Consider either accepting this or doing a two stage update. First,
the pool is updated with a zero distribution amount simply for the
harvest. After the balance and rewardDebt are adjusted, a second
update occurs to distribute the fees.

Resolution

⚫ ACKNOWLEDGED

Issue #49	**The fee of the first deposit does not get added towards the `erc20Reward`**
Severity	🟡 LOW SEVERITY
Location	<u>Lines 250-279</u>

```
function updatePoolWithFee(
    uint256 _pid,
    uint256 _depositFee
)
internal
{
    PoolInfo storage pool = poolInfo[_pid];
    uint256 lastTimestamp = block.timestamp < endTimestamp ?
block.timestamp : endTimestamp;

    if (lastTimestamp <= pool.lastRewardTimestamp) {
        lastTimestamp = pool.lastRewardTimestamp;
    }

    uint256 lpSupply = pool.totalDeposits;

    if (lpSupply == 0) {
        pool.lastRewardTimestamp = lastTimestamp;
        return;
    }

    uint256 nrOfSeconds =
lastTimestamp.sub(pool.lastRewardTimestamp);

    // Add to the reward fee taken, and distribute to all
users staking at the moment.
    uint256 reward = nrOfSeconds.mul(rewardPerSecond);
    uint256 erc20Reward =
reward.mul(pool.allocPoint).div(totalAllocPoint).add(_deposi
tFee);

    pool.accERC20PerShare =
pool.accERC20PerShare.add(erc20Reward.mul(1e36).div(lpSupply
));

    pool.lastRewardTimestamp = lastTimestamp;
}
```

AllocationStaking

Paladin Blockchain Security

Description	Due to the logical issue, there will be no distribution of the first deposit fee towards the `erc20Reward` — the fee will simply be stuck in the contract. If the `deposit` function on pool zero gets called it will also call the `updatePoolWithFee` function.
	This function is responsible for calculating the `accERC20PerShare`. However, due to the fact that the `lpSupply` is zero before the first deposit, it will return before the fee was added to `erc20Reward`. This will result in a loss of the mentioned fee.
Recommendation	Consider whether this scenario is acceptable, and if not, consider handling this scenario more explicitly.
Resolution	● ACKNOWLEDGED

Issue #50	**burnFromUser does not trigger within a deposit if withdrawalFeePending is greater than zero and withdrawalFeeDepositAmount is zero**
Severity	● LOW SEVERITY
Location	Lines 404-406 `if(withdrawalFeeDepositAmount > 0) {` `    // Update accounting around burns` `    burnFromUser(msg.sender, _pid,` `withdrawalFeeDepositAmount.add(withdrawalFeePending));`
Description	The `burnFromUser` call within `deposit` can burn two different amounts: the `withdrawalFeeDepositAmount` and the `withdrawalFeePending`. The `if` statement however only causes the burn to go through if the first amount is greater than zero. If this amount were to be zero while the second amount is non-zero, the burn would not occur.
Recommendation	Consider whether this is desired. If not, consider updating the if statement to include an or that the second amount must be non-zero.
Resolution	● ACKNOWLEDGED

Issue #51	The pending function will revert if `totalAllocPoint` is zero
Severity	🟡 LOW SEVERITY
Description	In the pending 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 `lastRewardTimestamp` if the `totalAllocPoint` variable is greater than zero. This check can simply be added to the existing check that verifies the `block.timestamp` and `lpSupply`, like so:
	`if (block.timestamp > pool.lastRewardTimestamp && lpSupply != 0 && totalAllocPoint > 0) {`
Resolution	⚫ ACKNOWLEDGED

AllocationStaking

Paladin Blockchain Security

Issue #52	Usage of encodePacked is discouraged in critical code sections
Severity	INFORMATIONAL

Location	<u>Lines 334, 360</u>
	`bytes32 nonceHash = keccak256(abi.encodePacked(functionName, nonce));`
	
	`abi.encodePacked(msg.sender, _pid, _amount, nonce)`

Description	The signature validation scheme checks the signature over a collection of bytes which is tightly packed. This is however not encouraged for critical sections of code as it could allow for hash collisions. This issue has been marked as informational as hash collisions are mainly an issue with variable length values (strings...) and the above code section does not have these. We therefore do not believe that there is any way to abuse this hash but would still like to recommend the best practice which more effectively guarantees this. It should be noted that encodePacked is at some point used with a variable length string, which is discouraged.
Recommendation	Consider using `abi.encode` instead of `abi.encodePacked`.
Resolution	ACKNOWLEDGED

Issue #53	safeTransfer should be used within the erc20Transfer function
Severity	● INFORMATIONAL
Location	Line 410
Description	In the erc20Transfer function, the transfer method is used to transfer tokens from the contract to an external address. This will not work for tokens that will return false on transfer (or malformed tokens that do not have a return value).
Recommendation	Consider using safeTransfer instead of transfer as is done throughout most of this contract.
Resolution	● ACKNOWLEDGED

Issue #54	SafeMath is not used
Severity	● INFORMATIONAL
Location	Lines 129, 476
Description	Using raw addition or subtraction instead of SafeMath can result in underflow / overflow.
	This issue has been marked as informational given that the locations are non-critical and overflow is unlikely.
Recommendation	Consider using SafeMath as in the rest of the contract.
Resolution	● ACKNOWLEDGED

Issue #55	Certain functions can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword: ⁻ `fund` ⁻ `add` ⁻ `setDepositFee` ⁻ `set` ⁻ `pending` ⁻ `deposit` ⁻ `withdraw` ⁻ `compound` ⁻ `emergencyWithdraw` ⁻ `setPostSaleWithdrawPenaltyPercentAndLength` Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the aforementioned variables as external.
Resolution	ACKNOWLEDGED

Issue #56	**salesRegistered can only be viewed on the contract in the userInfo struct, however it is not possible to view it on the front-end**
Severity	INFORMATIONAL
Description	It is impossible to view `salesRegistered` on the front-end since there is no `view` function for the variable.
Recommendation	Consider either removing the array or adding a `view` function.
Resolution	ACKNOWLEDGED

Issue #57	Lack of events for certain functions
Severity	🟣 INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications: ⁻ `setSalesFactory` ⁻ `fund` ⁻ `add` ⁻ `set` ⁻ `setTokensUnlockTime` ⁻ `redistributeXava` ⁻ `setPostSaleWithdrawPenaltyPercentAndLength` ⁻ `setAdmin`
Recommendation	Add events for the above functions.
Resolution	⚫ ACKNOWLEDGED

Issue #58	**Lack of validation: `startTimestamp` should be in the future; add function has no check for existing tokens**
Severity	● INFORMATIONAL

Description	<u>Lines 91-112</u>

```
function initialize(
        IERC20 _erc20,
        uint256 _rewardPerSecond,
        uint256 _startTimestamp,
        address _salesFactory,
        uint256 _depositFeePercent,
        uint256 _depositFeePrecision
    )
    initializer
    public
{
    __Ownable_init();
    erc20 = _erc20;
    rewardPerSecond = _rewardPerSecond;
    startTimestamp = _startTimestamp;
    endTimestamp = _startTimestamp;
    // Create sales factory contract
    salesFactory = ISalesFactory(_salesFactory);

    setDepositFeeInternal(_depositFeePercent,
_depositFeePrecision);
}
```

During the contract initialization, `startTimestamp` can be set to any timestamp. It should be only set to a timestamp which is in the future.

```
function add(uint256 _allocPoint, IERC20 _lpToken, bool
_withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardTimestamp = block.timestamp >
startTimestamp ? block.timestamp : startTimestamp;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    // Push new PoolInfo
    poolInfo.push(
        PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardTimestamp: lastRewardTimestamp,
            accERC20PerShare: 0,
            totalDeposits: 0
        })
    );
}
```

Governance has the ability to add new pools using the add function. However, there is no check if the _lpToken is an actual ERC20 token.

Recommendation	Consider adding require(_startTimestamp >= block.timestamp) to ensure the timestamp will be in the future.
	Consider adding _lpToken.balanceOf(address(this)); to the beginning of the function.
Resolution	● ACKNOWLEDGED

Issue #59	Unnecessary use of `address(msg.sender)`
Severity	● INFORMATIONAL
Location	Lines 128, 323, 400, 464
Description	It is not necessary to wrap `msg.sender` with `address()`.
Recommendation	Consider using `msg.sender` throughout the contract.
Resolution	● ACKNOWLEDGED

2.9   AvalaunchBadgeFactory

The AvalaunchBadgeFactory is an ERC-1155 NFT contract where the governance can define different badges with different multipliers using createBadges. Each badge they can then mint to multiple users using mintBadges. The audit scope does not contain more details about how these badges will be used.

2.9.1   Privileged Roles

The following functions can be called by the owner:

- pause

- unpause

- setNewUri

- setNewContractUri

- createBadges

- mintBadges

2.9.2   Issues & Recommendations

Issue #60	**mintBadges mint receipt hook has an outdated badgeIdToMintedSupply which can be a cause of exploits in derivative contracts**
Severity	🔴 MEDIUM SEVERITY

Location	Lines 145-149
	
	`_mint(receivers[i], badgeIds[i], 1, "0x0");`
	`emit BadgeMint(badgeIds[i], receivers[i]);`
	
	`// Increase total minted supply`
	`badgeIdToMintedSupply[badgeIds[i]] =`
	`badgeIdToMintedSupply[badgeIds[i]].add(1);`

Description	Within the `mint` function of the badge contract, the badge supply is updated after `_mint` is called. However, during `_mint`, the `onERC1155Received` function is called on the potentially malicious receiver. At this point of time, the malicious receiver can reenter into any part of the contract or the whole Avalaunch system at large. This in itself is not an issue. However, since `badgeIdToMintedSupply` has not been incremented yet, there is an inconsistency at this point as the receiver has already received their tokens. This could specifically cause exploits if the badge supply is used for critical functionalities in the derivative contracts outside of this audit scope. This vulnerability used to be present in the ERC-1155 `totalSupply` extension by OpenZeppelin. It was reported by ChainSecurity and their description of the issue can be read here. This issue has caused significant panic for certain protocols where the supply had significant importance in derivative contracts and where a discrepancy like this could cause exploitation. As there were no derivative contracts included within the audit scope, this issue is marked as medium severity.
Recommendation	Consider using the patched ERC1155Supply extension by OpenZeppelin. Moving the supply addition to before the `mint` would not really be a sufficient fix as the `_beforeTokenTransfer` hook would be inconsistent at that point.

Resolution ✅ **RESOLVED**

This issue was resolved similarly to how OpenZeppelin resolved this issue when it was disclosed within `ERC1155Supply`. Specifically, the increment of the minted supply has been moved to the `_beforeTokenTransfer` hook.

Issue #61	Certain functions can be made external

Severity 🟣 INFORMATIONAL

Description

Functions that are not used within the contract but only externally can be marked as such with the external keyword:

- `pause`
- `unpause`
- `setNewUri`
- `setNewContractUri`

Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.

Recommendation Consider marking the above functions as external.

Resolution ✅ RESOLVED

Issue #62	Lack of events for certain functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications: - `pause` - `unpause` - `setNewUri` - `setNewContractUri`
Recommendation	Add events for the above functions.
Resolution	PARTIALLY RESOLVED As `setNewUri` and `setNewContractUri` emit the same event, which is generally considered bad practice, this issue is left partially open.

Issue #63	Gas optimization: Usage of uint32 has causes extra gas usage
Severity	INFORMATIONAL
Description	Within a for loop in the contract, uint32 is used for the indices. This has no advantage over using uint256 as the word size is 256 bits within ethereum. In fact, if you were to do a gas usage comparison, uint256 would turn out being cheaper as less conversions are necessary.
Recommendation	Consider using uint256 consistently. The only argument for smaller data types is to pack them into structs. Further gas optimizations can be made by replacing the `memory` parameters with `calldata`.
Resolution	RESOLVED `uint32` was changed to `uint`.