Security Audit

Full Audit Report for Valletoken [With Re-assesment]

Client: Valletoken
https://valletoken.com/

July 9, 2026

Version: 1.2

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Table of Contents

1 Disclaimer 3
2 Security Assessment Procedure 4
3 Risk Rating 5
4 Category 6
5 Executive Summary 7
6 Audit Information 8
7 Findings List 13
8 Findings 14

HIGH	DOS-01	Denial of Service (DoS) on Final Stage Sell-out	14
MEDIUM	SEC-01	Arithmetic Underflow / Panic in `claimableOf()` due to `releaseTime` modification	16
LOW	SEC-02	Early Exit Penalty Only Slashes Rewards, Not Principal	18
LOW	SEC-03	Precision Truncation & Unclaimable Reward Dust via Frequent Claims	20
LOW	SEC-04	Stage Duration Expansion when advancing stages early	22
INFO	CEN-01	Centralization Risk	24
INFO	SEC-05	Capped Accumulation / Expiration for Flexible Tier Staking	28
INFO	SNC-01	Conformity to Solidity naming conventions	29
INFO	SEC-06	Allocation Precision Dust and Expensive Storage Reads	30
INFO	UCE-01	Use Custom Errors	32
INFO	LRS-01	Use of Long Revert Strings in `require()` or `revert()`	34
INFO	UIC-01	Use `!= 0` instead of `> 0` for unsigned integer comparison	35
INFO	UBS-01	Using bools for storage incurs overhead	36
INFO	DIV-01	Don't initialize variables with default value	37
INFO	PDM-01	Use of Post-Increment `i++` Instead of Pre-Increment `++i` in Loops	38
INFO	IRU-01	Inefficient `require()` Usage with && Conditions	39
INFO	CAL-01	Uncached Array Length Access in Loop	40

A Appendix 41
A.1 UML Class Diagram 41
A.2 Contracts Description Table 42
A.3 Components / Dependencies 45
A.4 Source Units in Scope 47

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

1 Disclaimer

This security assessment does not guarantee the security of the program instruction received from the client, herein referred to as “Source code.”

The Service Provider will not be held liable for any legal liability arising from errors in the security assessment. The responsibility for ensuring the security of the program instruction will be solely with the Client, herein referred to as “Service User.” The Service User agrees not to hold the Service Provider liable for any such liability.

By contract, the Service Provider is committed to conducting security assessments with integrity, professional ethics, and transparency in delivering security assessments to users. The Service Provider reserves the right to postpone the delivery of the security assessment if necessary, regardless of the reason for the delay. The Service Provider will not be held responsible for any delayed security assessments.

If the Service Provider identifies a vulnerability, we will notify the Service User via a Preliminary Report, which will be maintained in confidence for security purposes. The Service Provider disclaims responsibility in the event of any attacks occurring before or after conducting a security assessment. All responsibility for ensuring the security of the program instruction will be solely with the Service User.

Please be advised that the Security Assessment is not an investment advisory document and should not be interpreted as such. We hereby disclaim any liability arising from any investment:

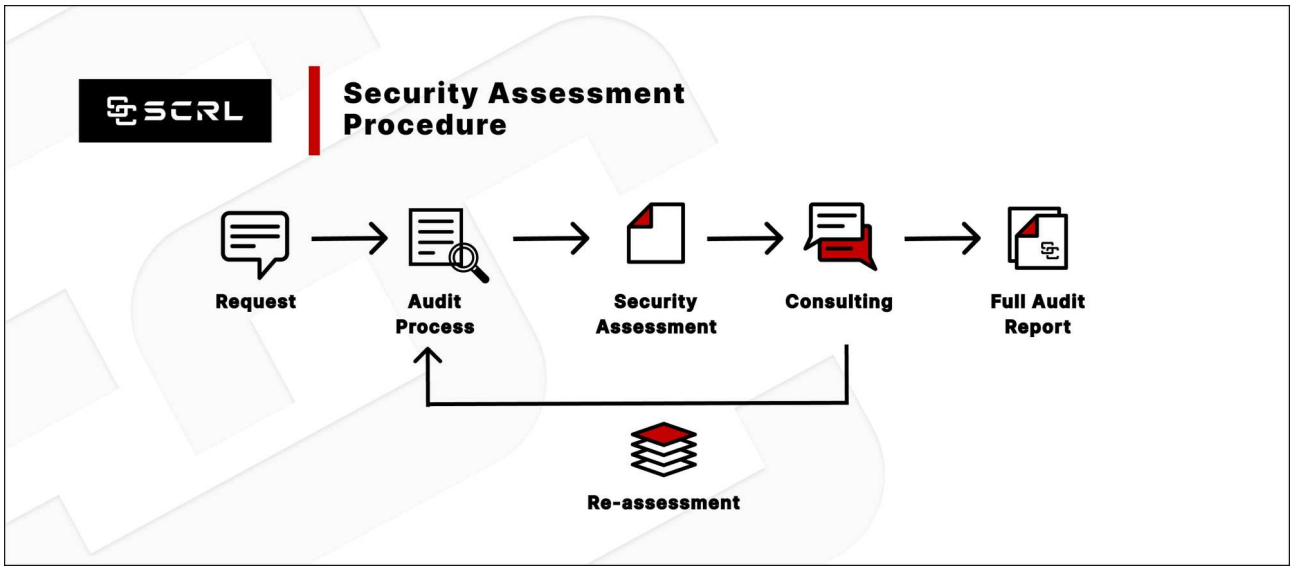
- Security Assessment Is **Not Financial/Investment Advice** Any loss arising from any investment in any project is the responsibility of the investor.
- SCRL **disclaims** any liability incurred, whether it's **Rugpull, Abandonment, Soft Rugpull, Exploit, Exit Scam.**
- **Cryptocurrency and digital token involve high risks; investors may lose all investment money and should study information carefully and make investments according to own risk profile.**
- Please thoroughly comprehend any aspect of a project, including liquidity checks and pre-sale stages, and assume the associated risks independently. We hereby disclaim any liability for any investment decisions made by you.

For full Legal / TOS / Privacy Policy please visit: https://scrl.io/legal

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

2 Security Assessment Procedure



- **Request:** The client must submit a formal request and follow the procedure. By submitting the source code and agreeing to the terms of service.
- **Audit Process:** Check for vulnerabilities and vulnerabilities from source code obtained by experts using formal verification methods, including using powerful tools such as Static Analysis, SWC Registry, Dynamic Security Analysis, Automated Security Tools, CWE, Syntax & Parameter Check with AI ,WAS (Warning Avoidance System a python script tools powered by SCRL).
- **Security Assessment:** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
- **Consulting:** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
- **Re-assessment:** Reassess the security when the client implements the source code improvements and if the client is satisfied with the results of the audit. We will proceed to the next step.
- **Full Audit Report:** SCRL provides clients with official security assessment reports informing them of risks and vulnerabilities. Officially and it is assumed that the client has been informed of all the information.

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

3 Risk Rating

Risk rating using this commonly defined: **Risk rating = impact * confidence**

- **Impact:** The severity and potential impact of an attacker attack
- **Confidence:** Ensuring that attackers expose and use this vulnerability

Confidence & Impact [Likelihood]	Low	Medium	High
Low	**Informational**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods, Risk rating = impact * confidence It is categorized into 7 categories severity based

Severity Risk	Description
Critical	Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
High	High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.
Medium	It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
Low	While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.
Informational	Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide additional information, recommendations

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

4 Category

Category	Description
Centralization	Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission
Economics Risk	Economics Risk is Risks that may affect the economic mechanism system, such as the ability to increase Mint token
Logical Issue	Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.
Authorization	Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.
Mathematical	Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.
Naming Conventions	naming variables that may affect code understanding or naming inconsistencies
Security Risk	Security Risk of loss or damage if it's no mitigate
Coding Style	Coding Style is Tips coding for efficiency performance
Best Practices	Best Practices is suggestions for improvement
Optimization	Optimization is performance improvement
Gas Optimization	Gas Optimization is increase performance to avoid expensive gas
Dead Code	Dead Code having unused code This may result in wasted resources and gas fees.
Input Validation	Proper input validation is essential to ensure that a smart contract processes only valid and anticipated data. Failure to implement robust input validation mechanisms may lead to various security vulnerabilities, including logic manipulation, unauthorized access, and unintended contract behavior.

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

5 Executive Summary

For this security assessment, SCRL received a request on **June 27, 2026**

Audit Score	Project Website	X (Twitter)	Telegram
9.3	https://valletoken.com/	https://x.com/valletoken_	https://t.me/vallecapitaltoken

Client	Confidential	Audit Method	Language
Valletoken	Public	Static Analysis + Dynamic Analysis + Manual Review + Formal Verification + Hyper Fuzzing + WAS (Custom Detector)	Solidity

Vulnerabilities Summary:

17	7	1	9	0
Total Findings	**Resolved**	**Partially-resolve**	**Acknowledge**	**Declined**

0	Critical		Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
1	High	Resolved: 1	High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users funds and data.
1	Medium	Resolved: 1	It essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
3	Low	Resolved: 3	While low-severity issues can be less urgent, it still advisable to address them to improve the overall security posture of the smart contract.
12	Informational	Resolved: 2 Partially resolve: 1 Acknowledge: 9	Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide information

SCRL
© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

6 Audit Information

Audit Scope:

File	SHA1-Hash
src/VCTToken.sol	934261eaa380211882f6b85192dd0baeaa1aceff
src/VCTPresaleV2.sol	c33b5f62dcdad57def8f268840ceeb2c8ebf39db
src/VCTStakingV2.sol	c623aab53d5bc48ce17958c30ec28d32047f50c0

Re-assessment Scope The Valletoken Team has been request for re-assesment on [2026-07-02] by updated their github code https://github.com/arandix/vct-presale-audit/ with this 4 commit updated 525e93b75e386bd701e9939f396a0d344846d617, 7e5cead6c5303d774eab764791ab8861465da515, 4d1750eb6a4ca1245da5d09c92c97205aea44f9a, dc692d19e079312bd04bd084c06156e50f6d539a

File	SHA1-Hash
src/VCTPresaleV2.sol	e2555bfc5c091a162106b0ba673484d710eb611f
src/VCTStakingV2.sol	d6c800a93f3b4cdec575c652122458ededfa53fd

Second Re-assessment Scope The Valletoken Team has been requested for reassessment on [2026-07-07] after being re-deployed and resolving some issues with this deployed contract

Chain	Contract Address
BNB Chain	0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f (VCTPresaleV2)
BNB Chain	0x4cf0cf09960AD24eA97Ae1Bd969200D578567BcE (VCTStakingV2)

Onchain Scope:

Chain	Contract Address
BNB Chain	0x032d721a14041479ca7f3bf8cab1afc694ad42fa (Token Contract)
BNB Chain	0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f (VCTPresaleV2 Contract)
BNB Chain	0x4cf0cf09960AD24eA97Ae1Bd969200D578567BcE (VCTStakingV2 Contract)

Security Assessment Author:

Name	Role
Chinnakit J.	CEO & Founder
Ronny C.	CTO & Head of Security Researcher
Mark K.	Security Researcher

Smart Contract Audit Summary:

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Digital Sign:

ID: 035FD16F-D857-45FF-BD6C-AF47CF1D22BC
Digitally signed by <contact@scrl.io>
July 09, 2026 10:28 AM +07

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SWC Checklist

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Completed	No Risk
SWC-101	Integer Overflow and Underflow	Completed	No Risk
SWC-102	Outdated Compiler Version	Completed	No Risk
SWC-103	Floating Pragma	Completed	No Risk
SWC-104	Unchecked Call Return Value	Completed	No Risk
SWC-105	Unprotected Ether Withdrawal	Completed	No Risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Completed	No Risk
SWC-107	Reentrancy	Completed	No Risk
SWC-108	State Variable Default Visibility	Completed	No Risk
SWC-109	Uninitialized Storage Pointer	Completed	No Risk
SWC-110	Assert Violation	Completed	No Risk
SWC-111	Use of Deprecated Solidity Functions	Completed	No Risk
SWC-112	Delegatecall to Untrusted Callee	Completed	No Risk
SWC-113	DoS with Failed Call	Completed	No Risk
SWC-114	Transaction Order Dependence	Completed	No Risk
SWC-115	Authorization through tx.origin	Completed	No Risk
SWC-116	Block values as a proxy for time	Completed	No Risk
SWC-117	Signature Malleability	Completed	No Risk
SWC-118	Incorrect Constructor Name	Completed	No Risk
SWC-119	Shadowing State Variables	Completed	No Risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Completed	No Risk
SWC-121	Missing Protection against Signature Replay Attacks	Completed	No Risk

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

ID	Title	Scanning	Result
SWC-122	Lack of Proper Signature Verification	Completed	No Risk
SWC-123	Requirement Violation	Completed	No Risk
SWC-124	Write to Arbitrary Storage Location	Completed	No Risk
SWC-125	Incorrect Inheritance Order	Completed	No Risk
SWC-126	Insufficient Gas Griefing	Completed	No Risk
SWC-127	Arbitrary Jump with Function Type Variable	Completed	No Risk
SWC-128	DoS With Block Gas Limit	Completed	No Risk
SWC-129	Typographical Error	Completed	No Risk
SWC-130	Right-To-Left-Override control character (U+202E)	Completed	No Risk
SWC-131	Presence of unused variables	Completed	No Risk
SWC-132	Unexpected Ether balance	Completed	No Risk
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Completed	No Risk
SWC-134	Message call with hardcoded gas amount	Completed	No Risk
SWC-135	Code With No Effects	Completed	No Risk
SWC-136	Unencrypted Private Data On-Chain	Completed	No Risk

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

OWASP Smart Contract Top 10 2025 Checklist

ID	Title	Scanning	Result
SC01:2025	Access Control Vulnerabilities	Completed	No Risk
SC02:2025	Price Oracle Manipulation	Completed	No Risk
SC03:2025	Logic Errors	Completed	No Risk
SC04:2025	Lack of Input Validation	Completed	No Risk
SC05:2025	Reentrancy Attacks	Completed	No Risk
SC06:2025	Unchecked External Calls	Completed	No Risk
SC07:2025	Flash Loan Attacks	Completed	No Risk
SC08:2025	Integer Overflow and Underflow	Completed	No Risk
SC09:2025	Insecure Randomness	Completed	No Risk
SC10:2025	Denial of Service (DoS) Attacks	Completed	No Risk

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

7 Findings List

0	1	1	3	12
Critical	High	Medium	Low	Informational

This security assessment report for **Valletoken** a total of **17 vulnerabilities.** The evaluation was conducted using the **Static Analysis + Dynamic Analysis + Manual Review + Formal Verification + Hyper Fuzzing + WAS (Custom Detector)** security assessment methodology.

ID	Vulnerabilities	Severity	Category	Status
DOS-01	Denial of Service (DoS) on Final Stage Sell-out	High	Security Risk	Resolved
SEC-01	Arithmetic Underflow / Panic in `claimableOf()` due to `releaseTime` modification	Medium	Security Risk	Resolved
SEC-02	Early Exit Penalty Only Slashes Rewards, Not Principal	Low	Logical Issue	Resolved
SEC-03	Precision Truncation & Unclaimable Reward Dust via Frequent Claims	Low	Logical Issue	Resolved
SEC-04	Stage Duration Expansion when advancing stages early	Low	Logical Issue	Resolved
CEN-01	Centralization Risk	Informational	Centralization	Partially resolve
SEC-05	Capped Accumulation / Expiration for Flexible Tier Staking	Informational	Best Practices	Acknowledge
SNC-01	Conformity to Solidity naming conventions	Informational	Coding Style	Resolved
SEC-06	Allocation Precision Dust and Expensive Storage Reads	Informational	Best Practices	Resolved
UCE-01	Use Custom Errors	Informational	Gas Optimization	Acknowledge
LRS-01	Use of Long Revert Strings in `require()` or `revert()`	Informational	Gas Optimization	Acknowledge
UIC-01	Use `!= 0` instead of `> 0` for unsigned integer comparison	Informational	Gas Optimization	Acknowledge
UBS-01	Using bools for storage incurs overhead	Informational	Gas Optimization	Acknowledge
DIV-01	Don't initialize variables with default value	Informational	Gas Optimization	Acknowledge
PDM-01	Use of Post-Increment `i++` Instead of Pre-Increment `++i` in Loops	Informational	Gas Optimization	Acknowledge
IRU-01	Inefficient `require()` Usage with && Conditions	Informational	Gas Optimization	Acknowledge
CAL-01	Uncached Array Length Access in Loop	Informational	Gas Optimization	Acknowledge

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

8 Findings

DOS-01: Denial of Service (DoS) on Final Stage Sell-out

Vulnerabilities	Severity	Category	Status
Denial of Service (DoS) on Final Stage Sell-out	High	Security Risk	Resolved

Finding Code

File: `VCTPresaleV2.sol`

```
357:         if (s.sold >= s.allocation) {
358:             _advanceStage();
359:         }
```

Description

In `_recordPurchase()`, whenever a purchase causes a stage to sell out (`s.sold >= s.allocation`), the contract automatically calls `_advanceStage()`. However, `_advanceStage()` contains the check `require(currentStage < STAGE_COUNT - 1, "Presale: all stages complete")`. If a buyer purchases the last remaining tokens in the final stage (Stage 14 / Round 15), the transaction will revert because it attempts to advance past the final stage.

Impact: The final stage can never be 100% sold out. Any transaction that attempts to buy the remaining allocation of the last stage will revert, permanently locking the final tokens.

SCRL provides further explanation. This problem prevents the final pre-sale stage from achieving 100% completion. Trading is still possible, but a fractional percentage will remain, for example, 99.99% leaving 0.01%.

For example, if someone has already purchased 98% of the tokens in the final stage round, and another user wants to purchase the remaining 2%, that transaction will revert.

However, this issue only occurs during the final pre-sale and only affects users who want to purchase the remaining tokens to complete the 100% sale in that final stage round.

Recommendation

Wrap the `_advanceStage()` call inside `_recordPurchase()` with a check to ensure it is not the final stage.

```
        if (s.sold >= s.allocation) {
            if (currentStage < STAGE_COUNT - 1) {
                _advanceStage();
            }
        }
```

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit 525e93b75e386bd701e9939f396a0d344846d617, but has not yet re-deployed the contract to fix it. If you continue using Contract Address: 0x38895479fbf79AF5F78Bc2d5e236057809a61E3b, you will still encounter this issue.

The Valletoken Team has applied this fix:

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Inside `_recordPurchase()`, the contract now verifies if a next stage exists before calling `_advanceStage()`.

```
if (s.sold >= s.allocation && currentStage < STAGE_COUNT - 1) {
    _advanceStage();
}
```

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTPresaleV2" Contract to BNB Mainnet at 0x282ee223b6aa4b019408952486a9648cee8f76d84cef4533f58f787712581fbc with Contract Address 0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f and resolved this issue.

References

SC10:2025 - Denial of Service (DoS) Attacks: DoS attacks exploit vulnerabilities to exhaust contract resources, rendering it non-functional. Examples include excessive gas consumption in loops or function calls designed to disrupt normal contract operation.

https://owasp.org/www-project-smart-contract-top-10/2025/en/src/SC10-denial-of-service.md

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SEC-01: Arithmetic Underflow / Panic in `claimableOf()` due to `releaseTime` modification

Vulnerabilities	Severity	Category	Status
Arithmetic Underflow / Panic in `claimableOf()` due to `releaseTime` modification	Medium	Security Risk	Resolved

Finding Code

File: `VCTPresaleV2.sol`

```
219:         return vestedOf(buyer) - purchases[buyer].claimedAmount;
```

Description

The owner can call `setReleaseTime()` at any time. If the owner delays the `releaseTime` after users have already claimed their initial TGE tokens, the output of `vestedOf(buyer)` will decrease (or become 0). When `claimableOf()` calculates `vestedOf(buyer) - purchases[buyer].claimedAmount`, it will result in an arithmetic underflow (Panic error in Solidity 0.8.x).

Impact: Users' ability to view their balances or interact with the contract (e.g. calling `claimableOf()` via frontends) will be completely broken (DoS) due to the underflow revert until the new time conditions are met.

Recommendation

Prevent changing the `releaseTime` once the TGE has started.

```
    function setReleaseTime(uint256 _releaseTime) external onlyOwner {
        require(block.timestamp < releaseTime, "Presale: claims already started");
        ...
```

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit 7e5cead6c5303d774eab764791ab8861465da515, but has not yet re-deployed the contract to fix it. If you continue using Contract Address: 0x38895479fbf79AF5F78Bc2d5e236057809a61E3b, you will still encounter this issue.

The Valletoken Team has applied this fix:

Inside `setReleaseTime()`, the owner is restricted from changing the release time once claims have already started:

```
function setReleaseTime(uint256 newReleaseTime) external onlyOwner {
    require(block.timestamp < releaseTime, "Presale: claims already started");
    require(newReleaseTime > block.timestamp, "Presale: release in past");
    releaseTime = newReleaseTime;
    emit ReleaseTimeUpdated(newReleaseTime);
```

How ever the researcher has found that this vulnerability still exists. **[2026-07-03]**

Our test confirms that **a vulnerability (bypass) still exists** in `openClaimsNow()`.

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

The function `openClaimsNow()` allows the owner to shift the `releaseTime` to the current `block.timestamp` at any time, but it is **missing the safeguard check** (`require(block.timestamp < releaseTime, ...)`).

If the owner calls `openClaimsNow()` after claims have already started:

1. `releaseTime` is updated to the current `block.timestamp`.
2. This resets the vesting timeline, causing `elapsed` time (inside `vestedOf()`) to shrink to 0.
3. Users' vested balances drop back to just the TGE portion (10%).
4. For any user who has already claimed more than 10%, `claimableOf()` (`vestedOf() - claimedAmount`) underflows, causing all balance views and subsequent claim transactions to revert (DoS).

Our test successfully validates that calling `openClaimsNow()` after claims start triggers this underflow revert.

Recommendation

Restrict `openClaimsNow()` to only be called before claims start:

```
function openClaimsNow() external onlyOwner {
    require(block.timestamp < releaseTime, "Presale: claims already started");
    releaseTime = block.timestamp;
    emit ReleaseTimeUpdated(block.timestamp);
    emit ClaimsForceOpened();
}
```

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTPresaleV2" Contract to BNB Mainnet at 0x282ee223b6aa4b019408952486a9648cee8f76d84cef4533f58f787712581fbc with Contract Address 0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f and resolved this issue.

The Valletoken Team has applied this fix:

Both the `openClaimsNow()` bypass and the unguarded subtraction in `claimableOf()` have been resolved.

```
function openClaimsNow() external onlyOwner {
    require(block.timestamp < releaseTime, "Presale: claims already started");
    releaseTime = block.timestamp;
    emit ReleaseTimeUpdated(block.timestamp);
    emit ClaimsForceOpened();
}
```

Inside `claimableOf()`, the raw subtraction is replaced with a safe ternary guard to prevent arithmetic underflow:

```
function claimableOf(address buyer) public view returns (uint256) {
    uint256 vested  = vestedOf(buyer);
    uint256 claimed = purchases[buyer].claimedAmount;
    return vested > claimed ? vested - claimed : 0;
}
```

Assessment: Correct. Once `block.timestamp >= releaseTime`, `openClaimsNow()` reverts — the vesting timeline is frozen after TGE begins. The safe ternary in `claimableOf()` additionally ensures the function can never panic-revert regardless of state.

References

SC09:2026 - Integer Overflow and Underflow: Dangerous arithmetic on platforms or code paths without robust overflow checks, leading to wrapped values, broken invariants, and potential drains of liquidity or mis-accounting.

https://owasp.org/www-project-smart-contract-top-10/2026/en/src/SC09-integer-overflow-underflow.html

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SEC-02: Early Exit Penalty Only Slashes Rewards, Not Principal

Vulnerabilities	Severity	Category	Status
Early Exit Penalty Only Slashes Rewards, Not Principal	Low	Logical Issue	Resolved

Finding Code

File: `VCTStakingV2.sol`

```
133:         uint256 rewards = _pendingRewards(s);
134:         uint256 finalRewards = rewards;
135:         if (earlyExit && cfg.earlySlashBps > 0) {
136:             finalRewards = rewards - (rewards * cfg.earlySlashBps) / BASIS_POINTS; // slashed
stays in pool
137:         }
```

Description

Impact: The early exit mechanism in `unstake()` checks if the current time is before the lock expiry. If so, it applies `earlySlashBps` exclusively to the accrued rewards. The user's staked principal remains untouched and is returned 100% intact. This allows users to choose the highest APY tier (e.g. Bronze at 12% APY or Silver at 20% APY) instead of the Flexible tier (5% APY). Even if they exit early, their effective slashed APY can outperform the honest Flexible tier, making the Flexible tier economically obsolete. For example, a user staking in Bronze (12% APY, 50% slash) and exiting at Day 15 earns an effective APY of 6% over those 15 days, which is higher than the 5% APY they would have earned in the Flexible tier.

Recommendation

Apply the early exit penalty to the staked principal rather than the accrued rewards, or enforce a harsher penalty metric. For example:

```
        uint256 principal = s.amount;
        if (earlyExit && cfg.earlySlashBps > 0) {
            uint256 slashAmount = (principal * cfg.earlySlashBps) / BASIS_POINTS;
            principal -= slashAmount;
        }
```

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit dc692d19e079312bd04bd084c06156e50f6d539a.

The Valletoken Team has applied this fix:

The team modified the early exit branch in `unstake()` (lines 136-137):

```
uint256 rewards = _pendingRewards(s);
uint256 finalRewards = earlyExit ? 0 : rewards;
```

If a user exits early, their *pending* rewards are forced to `0`.

How ever the researcher has found that this vulnerability still exists. **[2026-07-03]**

Our test confirms a **vulnerability (bypass) still exists** exists.

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Because `claimRewards()` does not check if the stake is currently locked, a user can execute the exploit in two steps:

1. **Call `claimRewards()`** to withdraw all accrued rewards at the higher APY rate (e.g., 12% APY for Bronze).
2. **Call `unstake()`** to exit early. Since pending rewards are reset to 0 in step 1, `finalRewards` becomes `0`. However, the user already has the claimed rewards in their wallet, and `unstake()` returns 100% of their principal.

As a result, the early exit penalty is completely bypassed.

Recommendation

To properly fix this issue, either:

1. **Slash the principal**:
```
uint256 principal = s.amount;
if (earlyExit && cfg.earlySlashBps > 0) {
    uint256 slashAmount = (principal * cfg.earlySlashBps) / BASIS_POINTS;
    principal -= slashAmount;
}
```
2. **Restrict reward claiming before lock expiry**: If claims are forbidden until `unlockTime` is reached, then the user cannot exit early with claimed rewards. However, slashing the principal remains the industry standard and most robust solution.

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTStakingV2" Contract to BNB Mainnet at 0xe9ad7d4abaaa32915f3fb8ecd5574fa6cd7dac201aaec957476a7c88432d6d01 with Contract Address 0x4cf0cf09960AD24eA97Ae1Bd969200D578567BcE and resolved this issue.

The Valletoken Team has applied this fix:

The `claimRewards()` bypass identified in Re-assignment 1 has been resolved.

Applied Fix

Inside `claimRewards()`, a lock expiry check is added so that rewards cannot be claimed before the stake's unlock time is reached:

```
function claimRewards(uint256 stakeId) external nonReentrant whenNotPaused {
    Stake storage s = _getActiveStake(msg.sender, stakeId);
    // SEC-02 fix: rewards cannot be claimed before the lock expires. Together with the
    // early-exit reward forfeit in unstake(), this closes the bypass where a locked staker
    // claimed high-APY rewards mid-lock and then exited early with 100% principal.
    // Flexible tier has unlockTime == startTime, so it is unaffected.
    require(block.timestamp >= s.unlockTime, "Staking: locked until unlock");
    uint256 rewards = _pendingRewards(s);
    require(rewards > 0, "Staking: no rewards");
    ...
}
```

Assessment: Correct. Locked stakers cannot call `claimRewards()` until their `unlockTime` is reached, closing the two-step drain-then-exit bypass. The Flexible tier (`unlockTime == startTime`) is unaffected.

References

SC02:2026 - Business Logic Vulnerabilities: Design-level flaws in lending, AMM, reward, or governance logic that break intended economic or functional rules, enabling attackers to extract value even when low-level checks appear correct.

https://owasp.org/www-project-smart-contract-top-10/2026/en/src/SC02-business-logic-vulnerabilities.html

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SEC-03: Precision Truncation & Unclaimable Reward Dust via Frequent Claims

Vulnerabilities	Severity	Category	Status
Precision Truncation & Unclaimable Reward Dust via Frequent Claims	Low	Logical Issue	Resolved

Finding Code

File: `VCTStakingV2.sol`

```
195: uint256 elapsed = t - s.lastClaimTime;
196: uint256 r = (s.amount _ cfg.apyBps _ elapsed) / (BASIS_POINTS * SECONDS_YEAR);
```

Description

Impact: The reward accrual calculation in `_pendingRewards()` performs linear integer division: `r = (s.amount * cfg.apyBps * elapsed) / (BASIS_POINTS * SECONDS_YEAR)`. Frequent claims lead to multiple truncations (floor divisions), leaving behind microscopic fractions of tokens (dust). Upon a final claim or unstake after `rewardEnd`, `s.lastClaimTime` is updated to `block.timestamp` (which is greater than `rewardEnd`). Due to the boundary check `if (t <= s.lastClaimTime) return 0;` (where `t` is capped at `rewardEnd`), any remaining precision dust (`s.projectedReward - s.rewardsPaid`) becomes permanently locked and unclaimable by the user.

Recommendation

When a user performs their final claim or unstakes after `rewardEnd`, award the remaining `projectedReward - rewardsPaid` directly instead of calculating elapsed time rewards. For example:

```
        if (block.timestamp >= rewardEnd) {
            return s.projectedReward - s.rewardsPaid;
        }
```

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit 4d1750eb6a4ca1245da5d09c92c97205aea44f9a.

The Valletoken Team has applied this fix:

The team added a boundary check at the beginning of `_pendingRewards()` (lines 197-199):

```
if (block.timestamp >= rewardEnd) {
    return remaining;
}
```

This bypasses elapsed time calculations when the stake duration is complete, directly returning the outstanding reserved reward `projectedReward - rewardsPaid`.

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTStakingV2" Contract to BNB Mainnet at 0xe9ad7d4abaaa32915f3fb8ecd5574fa6cd7dac201aaec957476a7c88432d6d01 with Contract Address 0x4cf0cf09960AD24eA97Ae1Bd969200D578567BcE and resolved this issue.

SCRL
© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

References

SC02:2026 - Business Logic Vulnerabilities: Design-level flaws in lending, AMM, reward, or governance logic that break intended economic or functional rules, enabling attackers to extract value even when low-level checks appear correct.

https://owasp.org/www-project-smart-contract-top-10/2026/en/src/SC02-business-logic-vulnerabilities.html

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SEC-04: Stage Duration Expansion when advancing stages early

Vulnerabilities	Severity	Category	Status
Stage Duration Expansion when advancing stages early	Low	Logical Issue	Resolved

Finding Code

File: `VCTPresaleV2.sol`

```
370:             if (ns.endTime < block.timestamp + STAGE_DURATION) {
371:                 ns.endTime = block.timestamp + STAGE_DURATION;
372:             }
```

Description

When a stage sells out early, it immediately starts the next stage. The contract attempts to cap the new endTime using `if (ns.endTime < block.timestamp + STAGE_DURATION)`. However, since the original `endTime` was initialized far in the future during deployment, this condition evaluates to false.

Impact: The `endTime` remains at its original scheduled timestamp. For example, if Stage 0 sells out on Day 1, Stage 1 will start immediately but will not end until Day 20, making Stage 1 last 19 days instead of the intended 10 days.

Recommendation

Strictly enforce the 10-day duration from the moment the new stage starts.

```
        if (ns.startTime > block.timestamp) {
            ns.startTime = block.timestamp;
            ns.endTime = block.timestamp + STAGE_DURATION;
        }
```

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit 4d1750eb6a4ca1245da5d09c92c97205aea44f9a.

The Valletoken Team has applied this fix:

The contract strictly assigns `ns.endTime` without the faulty check:

```
if (ns.startTime > block.timestamp) {
    ns.startTime = block.timestamp;
    ns.endTime = block.timestamp + STAGE_DURATION;
}
```

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTPresaleV2" Contract to BNB Mainnet at 0x282ee223b6aa4b019408952486a9648cee8f76d84cef4533f58f787712581fbc with Contract Address 0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f and resolved this issue.

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

References

SC02:2026 - Business Logic Vulnerabilities: Design-level flaws in lending, AMM, reward, or governance logic that break intended economic or functional rules, enabling attackers to extract value even when low-level checks appear correct.

https://owasp.org/www-project-smart-contract-top-10/2026/en/src/SC02-business-logic-vulnerabilities.html

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

CEN-01: Centralization Risk

Vulnerabilities	Severity	Category	Status
Centralization Risk	Informational	Centralization	Partially resolve

Finding Code

VCTStakingV2.sol:

```
29: contract VCTStakingV2 is ReentrancyGuard, Ownable2Step, Pausable {
71: constructor(address _vct) Ownable(msg.sender) {
156: function fundRewardPool(uint256 amount) external onlyOwner {
162: function withdrawExcessRewards(uint256 amount) external onlyOwner {
171: function pause() external onlyOwner { _pause(); }
172: function unpause() external onlyOwner { _unpause(); }
```

VCTPresaleV2.sol:

```
47: contract VCTPresaleV2 is ReentrancyGuard, Ownable2Step, Pausable {
125:     ) Ownable(msg.sender) {
234:     function distributeBatch(uint256 start, uint256 count) external onlyOwner nonReentrant {
255:     function advanceStage() external onlyOwner { _advanceStage(); }
267:     function setTreasury(address _treasury) external onlyOwner {
274:     function setReleaseTime(uint256 _releaseTime) external onlyOwner {
281:     function openClaimsNow() external onlyOwner {
288:     function sweepUnsoldVCT(uint256 amount) external onlyOwner {
296:     function pause()   external onlyOwner { _pause(); }
297:     function unpause() external onlyOwner { _unpause(); }
```

VCTToken.sol:

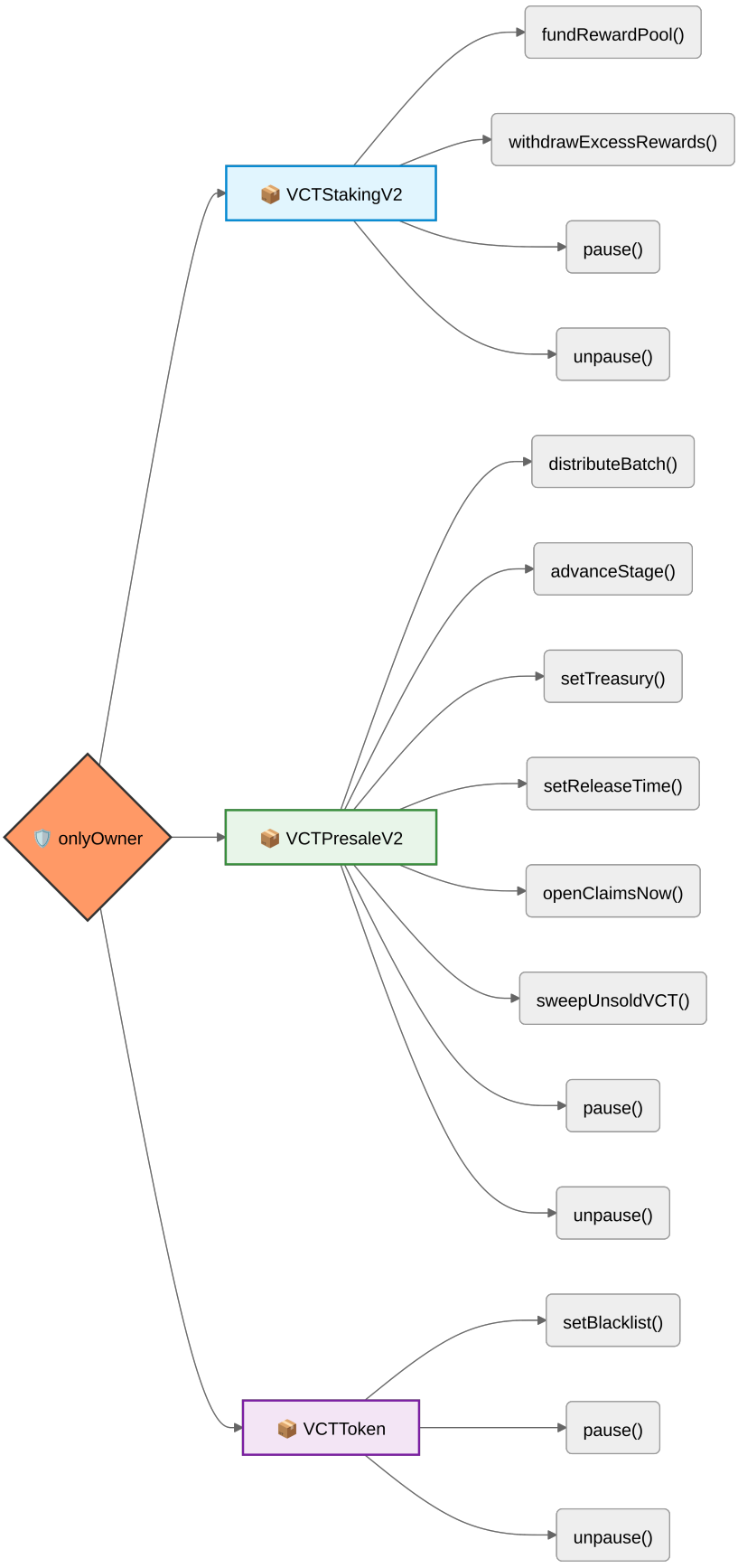
```
23: import {Ownable2Step, Ownable} from "@openzeppelin/contracts/access/Ownable2Step.sol";
49: contract VCTToken is ERC20, ERC20Pausable, Ownable2Step {
90:         Ownable(msg.sender)
163:     function setBlacklist(address account, bool status) external onlyOwner {
170:     function pause()   external onlyOwner { _pause();   }
173:     function unpause() external onlyOwner { _unpause(); }
```

Description

Centralization Risk

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC



A Centralization Risk has been identified for the `Contract` entity, which has privileges set to `onlyOwner`. This restriction allows the execution of the following functions: `function`

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Each function possesses the following capabilities:

VCTStakingV2.sol:

- `fundRewardPool`: Allows owner to transfer VCT tokens into the staking contract to fund the reward pool.
- `withdrawExcessRewards`: Allows owner to withdraw excess VCT tokens that are not currently staked and not reserved for rewards (cannot touch staker funds).
- `pause`: Allows owner to pause staking and claiming functionality.
- `unpause`: Allows owner to resume staking and claiming functionality.

VCTPresaleV2.sol:

- `distributeBatch`: Allows owner to automatically distribute vested tokens to buyers in batches.
- `advanceStage`: Allows owner to manually advance the presale to the next stage.
- `setTreasury`: Allows owner to change the multisig treasury address receiving all payments.
- `setReleaseTime`: Allows owner to adjust the TGE release time (cliff) when vesting starts.
- `openClaimsNow`: Allows owner to trigger TGE immediately, starting the 10% unlock and 12-month vesting schedule (emergency function).
- `sweepUnsoldVCT`: Allows owner to recover excess VCT tokens not owed to buyers.
- `pause`: Allows owner to pause purchasing tokens.
- `unpause`: Allows owner to resume purchasing tokens.

VCTToken.sol:

- `setBlacklist`: Allows owner to block or unblock an address from sending or receiving VCT.
- `pause`: Allows owner to pause all token transfers in emergency.
- `unpause`: Allows owner to resume token transfers.

Recommendation

In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

- Renounce the ownership and cannot turn back to claim a privileges to execution a function

OR

- Remove a contain centralization risk function and deployed a new contract

Alleviation

[2026-07-06]

The Valletoken Team has been updated to use "VCTTimelock" with 48-hours delay and will use a multisig. The `VCTTimelock` contract has been deployed at 0x06c7139134e0c2ba68d7a34ade47a84c612e23c4289ce5d443cb709ee78df073 on BNB Mainnet and using this contract address 0xc10a8f2B2B2aA10dF2541ed2DfF7fB541Ac29c71, SCRL has updated this issue status as "Partially Resolve"

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

References

--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SEC-05: Capped Accumulation / Expiration for Flexible Tier Staking

Vulnerabilities	Severity	Category	Status
Capped Accumulation / Expiration for Flexible Tier Staking	Informational	Best Practices	Acknowledge

Finding Code

File: `VCTStakingV2.sol`

```
191: uint256 rewardDuration = cfg.lockDuration == 0 ? SECONDS_YEAR : cfg.lockDuration;
192: uint256 rewardEnd = s.startTime + rewardDuration; // no rewards accrue past this point
```

Description

Impact: For the `FLEXIBLE` tier, `cfg.lockDuration` is set to `0`. In `_pendingRewards()`, the contract falls back to a 1-year duration: `uint256 rewardDuration = cfg.lockDuration == 0 ? SECONDS_YEAR : cfg.lockDuration;`. This hard-caps the reward accumulation period to exactly 365 days from the start of that stake. Flexible stakers stop earning rewards after 365 days and must manually unstake and restake to reset the timeline. This is an architectural constraint that should be clearly documented to manage user expectations.

Recommendation

Ensure this design constraint is clearly documented in user-facing materials and frontend interfaces so that flexible stakers know they need to renew their stakes annually.

Alleviation

--

References

--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SNC-01: Conformity to Solidity naming conventions

Vulnerabilities	Severity	Category	Status
Conformity to Solidity naming conventions	Informational	Coding Style	Resolved

Finding Code

```
   Variable `VCTPresaleV2.BONUS_BPS` (src/VCTPresaleV2.sol:75) is not in mixedCase
   Variable `VCTPresaleV2.BONUS_MIN` (src/VCTPresaleV2.sol:74) is not in mixedCase
   Parameter `VCTPresaleV2.setTreasury(address)._treasury` (src/VCTPresaleV2.sol:267) is not in
mixedCase
   Parameter `VCTPresaleV2.setReleaseTime(uint256)._releaseTime` (src/VCTPresaleV2.sol:274) is not in
mixedCase
```

Description

The smart contract does not fully adhere to Solidity's standard naming conventions. Consistent and conventional naming improves readability, maintainability, and helps avoid confusion or misinterpretation of a contract's intent.

Recommendation

It is recommended to refactor the contract to fully comply with Solidity's official naming conventions. This practice aligns with industry standards and facilitates more effective collaboration, auditing, and code comprehension.

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit dc692d19e079312bd04bd084c06156e50f6d539a, but has not yet re-deployed the contract to fix it. If you continue using Contract Address: 0x38895479fbf79AF5F78Bc2d5e236057809a61E3b, you will still encounter this issue.

The Valletoken Team has applied this fix:

- Removals: `BONUS_MIN` and `BONUS_BPS` state variables were completely removed (refactored into memory pure-comparisons).
- Renamed parameters in functions:

```
function setTreasury(address newTreasury) external onlyOwner { ... }
  function setReleaseTime(uint256 newReleaseTime) external onlyOwner { ... }
```

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTPresaleV2" Contract to BNB Mainnet at 0x282ee223b6aa4b019408952486a9648cee8f76d84cef4533f58f787712581fbc with Contract Address 0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f and resolved this issue.

References

Solidity Style Guide – Solidity Docs https://docs.soliditylang.org/en/latest/style-guide.html

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

SEC-06: Allocation Precision Dust and Expensive Storage Reads

Vulnerabilities	Severity	Category	Status
Allocation Precision Dust and Expensive Storage Reads	Informational	Best Practices	Resolved

Finding Code

File: `VCTPresaleV2.sol`

```
143:         uint256 alloc = PRESALE_ALLOCATION / STAGE_COUNT; // 227.5M ÷ 15 rounds
320:             if (usdValue6 >= BONUS_MIN[i]) bps = BONUS_BPS[i];
```

Description

1. **Precision Dust**: `227,500,000 * 1e18 / 15` leaves a remainder of 10 wei. This means the sum of all stage allocations is slightly less than `PRESALE_ALLOCATION`.
2. **Gas Optimization**: `BONUS_MIN` and `BONUS_BPS` are state variables (storage). The `_bonusFor()` loop reads from storage 4 times per purchase transaction, which is highly gas-inefficient.

Recommendation

1. Add the 10 wei remainder to the first stage's (Stage 0) allocation in the constructor.
2. Refactor `BONUS_MIN` and `BONUS_BPS` as constant getters or pure functions, and update `_bonusFor()` to use pure comparisons.

```
// Constructor remainder fix
stages[i] = Stage({
    ...
    allocation: alloc + (i == 0 ? remainder : 0),
    ...
```

```
// Gas optimization fix
function _bonusFor(uint256 usdValue6) internal pure returns (uint256) {
    if (usdValue6 >= 25_000 * 1e6) return 800;
    if (usdValue6 >= 10_000 * 1e6) return 500;
    if (usdValue6 >= 5_000 * 1e6) return 300;
    return 0;
```

Alleviation

[2026-07-02]

The Valletoken Team has updated the code on Github at commit dc692d19e079312bd04bd084c06156e50f6d539a, but has not yet re-deployed the contract to fix it. If you continue using Contract Address: 0x38895479fbf79AF5F78Bc2d5e236057809a61E3b, you will still encounter this issue.

The Valletoken Team has applied this fix:

1. The remainder is computed in the constructor and assigned to Stage 0:

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

```
uint256 remainder = PRESALE_ALLOCATION - (alloc * STAGE_COUNT);
...
allocation: alloc + (i == 0 ? remainder : 0)
```

2. The state arrays are removed and lookup logic is converted into a pure function `_bonusFor`:

```
function _bonusFor(uint256 usdValue6) internal pure returns (uint256) {
    if (usdValue6 >= 25_000 * 1e6) return 800;
    if (usdValue6 >= 10_000 * 1e6) return 500;
    if (usdValue6 >=  5_000 * 1e6) return 300;
    return 0;
}
```

[2026-07-06]

The Valletoken Team has been updated resolved code by deployed a "VCTPresaleV2" Contract to BNB Mainnet at 0x282ee223b6aa4b019408952486a9648cee8f76d84cef4533f58f787712581fbc with Contract Address 0xD1e2a1e4aEdd7316eCD4ae770c040da71B13dE1f and resolved this issue.

References

--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

UCE-01: Use Custom Errors

Vulnerabilities	Severity	Category	Status
Use Custom Errors	Informational	Gas Optimization	Acknowledge

Finding Code

VCTStakingV2.sol:

```
72: require(_vct != address(0), "Staking: zero token");
83: require(amount >= MIN_STAKE, "Staking: below minimum");
84: require(totalStaked[msg.sender] + amount <= MAX_STAKE, "Staking: exceeds wallet max");
88: require(rewardPool >= committedRewards + projected, "Staking: reward pool insufficient");
115: require(rewards > 0, "Staking: no rewards");
131: if (earlyExit) require(cfg.earlyExitAllowed, "Staking: lock not expired");
166: require(amount <= free, "Staking: would touch staker funds");
208: require(stakeId < stakes[user].length, "Staking: invalid stake id");
210: require(s.active, "Staking: stake not active");
```

VCTPresaleV2.sol:

```
126:         require(_vct      != address(0), "Presale: zero vct");
127:         require(_usdt     != address(0), "Presale: zero usdt");
128:         require(_usdc     != address(0), "Presale: zero usdc");
129:         require(_treasury != address(0), "Presale: zero treasury");
130:         require(_bnbFeed  != address(0), "Presale: zero feed");
131:         require(_startTime > block.timestamp, "Presale: start in past");
184:         require(msg.value > 0, "Presale: zero BNB");
193:         require(ok, "Presale: BNB transfer failed");
224:         require(claimsOpen(), "Presale: claims not open");
226:         require(sent > 0, "Presale: nothing to claim");
235:         require(claimsOpen(), "Presale: claims not open");
268:         require(_treasury != address(0), "Presale: zero address");
275:         require(_releaseTime > block.timestamp, "Presale: release in past");
292:         require(amount <= free, "Presale: would touch buyer tokens");
311:         require(price > 0, "Presale: invalid BNB price");
312:         require(block.timestamp - updatedAt <= 3600, "Presale: stale BNB price");
313:         require(answeredInRound >= roundId, "Presale: incomplete round");
333:         require(usdValue6 >= MIN_PURCHASE_USD, "Presale: below minimum");
334:         require(walletSpentUSD[msg.sender] + usdValue6 <= MAX_PURCHASE_USD, "Presale: exceeds
wallet max");
338:         require(lastPurchaseBlock[msg.sender] < block.number, "Presale: one tx per block");
344:         require(block.timestamp >= s.startTime && block.timestamp <= s.endTime, "Presale: stage
not active");
345:         require(s.sold + vctAmount <= s.allocation, "Presale: stage sold out");
363:         require(currentStage < STAGE_COUNT - 1, "Presale: all stages complete");
379:         revert("Presale: use buyWithBNB()");
```

VCTToken.sol:

```
92:         require(reserves.length > 0, "VCT: no reserves");
95:             require(reserves[i].account != address(0), "VCT: zero address in reserve");
96:             require(reserves[i].amount  > 0,           "VCT: zero amount in reserve");
116:         require(index < _reserves.length, "VCT: invalid reserve index");
133:         require(amount > 0, "VCT: zero burn amount");
```


© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

```
155:         require(current >= subtractedValue, "VCT: decreased allowance below zero");
164:         require(account != address(0), "VCT: zero address");
181:         require(!blacklisted[from], "VCT: sender blacklisted");
182:         require(!blacklisted[to],   "VCT: recipient blacklisted");
```

Description

The contract uses `require()` statements with revert strings to handle errors and enforce conditions. While this is syntactically valid, it is not the most efficient method in terms of gas consumption.

Starting with **Solidity 0.8.4**, developers can define and use custom errors as a more gas-efficient and structured alternative to revert strings. Custom errors reduce deployment and runtime costs by avoiding the storage of long string literals in contract bytecode.

Recommendation

Refactor require statements with revert strings into custom errors, particularly in frequently used functions or large codebases. This approach improves gas efficiency and allows for more structured error handling, enhancing the overall quality and professionalism of the code.

Alleviation

--

References

Control Structures – Solidity Docs https://docs.soliditylang.org/en/latest/control-structures.html#custom-errors

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

LRS-01: Use of Long Revert Strings in `require()` or `revert()`

Vulnerabilities	Severity	Category	Status
Use of Long Revert Strings in `require()` or `revert()`	Informational	Gas Optimization	Acknowledge

Finding Code

VCTStakingV2.sol:

```
88: require(rewardPool >= committedRewards + projected, "Staking: reward pool insufficient");
166: require(amount <= free, "Staking: would touch staker funds");
```

VCTPresaleV2.sol:

```
292:         require(amount <= free, "Presale: would touch buyer tokens");
```

VCTToken.sol:

```
155:         require(current >= subtractedValue, "VCT: decreased allowance below zero");
```

Description

The contract uses long revert strings in `require()` or `revert()` statements. While these strings can improve debugging during development, they increase the bytecode size and gas costs at both deployment and runtime.

In Ethereum, revert strings are stored in the bytecode and transmitted with the transaction when a revert occurs. Longer strings consume more storage and gas, which may lead to higher transaction fees and even deployment size limitations in extreme cases.

Recommendation

Long revert strings in the require statements consume more gas. It's more gas-efficient to use short revert strings or error codes. In addition, using custom errors can further optimize gas usage while providing clear and meaningful error messages.

Replace long revert strings with:

- Shorter, concise messages when strings are necessary.
- Preferably, use custom errors to achieve optimal gas efficiency while maintaining readability.

Alleviation

--

References

Solidity Docs – Custom Errors: https://docs.soliditylang.org/en/latest/control-structures.html#custom-errors

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

UIC-01: Use `!= 0` instead of `> 0` for unsigned integer comparison

Vulnerabilities	Severity	Category	Status
Use `!= 0` instead of `> 0` for unsigned integer comparison	Informational	Gas Optimization	Acknowledge

Finding Code

VCTStakingV2.sol:

```
115: require(rewards > 0, "Staking: no rewards");
135: if (earlyExit && cfg.earlySlashBps > 0) {
```

VCTPresaleV2.sol:

```
184:         require(msg.value > 0, "Presale: zero BNB");
226:         require(sent > 0, "Presale: nothing to claim");
311:         require(price > 0, "Presale: invalid BNB price");
```

VCTToken.sol:

```
92:         require(reserves.length > 0, "VCT: no reserves");
96:             require(reserves[i].amount  > 0,           "VCT: zero amount in reserve");
133:         require(amount > 0, "VCT: zero burn amount");
```

Description

The contract uses `> 0` to check whether an unsigned integer (uint) is non-zero. While functionally correct, this comparison is slightly less efficient than using `!= 0`, due to the way EVM opcodes are generated and optimized.

In the EVM, the `!=` operation is generally compiled to fewer instructions than `>`, resulting in minor gas savings. Since uint variables cannot be negative, both expressions yield the same result, but `!= 0` is considered a more gas-optimal and idiomatic choice for checking non-zero conditions.

Recommendation

Use `!= 0` instead of `> 0` when checking whether a `uint` is non-zero. This is a gas-efficient and stylistically preferred approach in Solidity.

Alleviation

--

References

--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

UBS-01: Using bools for storage incurs overhead

Vulnerabilities	Severity	Category	Status
Using bools for storage incurs overhead	Informational	Gas Optimization	Acknowledge

Finding Code

File: `VCTPresaleV2.sol`

```
93:     mapping(address => bool) private _isBuyer;
```

Description

Using bool types in storage variables can lead to **unnecessary gas overhead** due to the way the Ethereum Virtual Machine (EVM) handles storage writes. Although a bool only requires 1 byte, the EVM stores data in 32-byte storage slots. When a bool is modified, the entire 32-byte storage slot must be read and rewritten due to the need to preserve surrounding data.

This behavior results in **extra gas consumption**, especially when multiple bool variables are packed into the same slot or modified frequently. Additionally, slot packing may complicate upgradeability and reduce predictability of gas costs.

Impact:

- Increased gas usage during storage writes
- Potential complications in upgradeable contracts or proxy patterns
- Reduced performance for large-scale contracts or those with frequent state changes

Recommendation

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past.

Alleviation

--

References

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

DIV-01: Don't initialize variables with default value

Vulnerabilities	Severity	Category	Status
Don't initialize variables with default value	Informational	Gas Optimization	Acknowledge

Finding Code

VCTPresaleV2.sol:

```
145:         for (uint256 i = 0; i < STAGE_COUNT; i++) {
318:         uint256 bps = 0;
319:         for (uint256 i = 0; i < 4; i++) {
```

VCTToken.sol:

```
94:         for (uint256 i = 0; i < reserves.length; i++) {
```

Description

In Solidity, variables are automatically assigned default values when declared. For example, uint and int default to 0, bool defaults to false, and address defaults to 0x000...000. Explicitly assigning these default values during variable declaration is **redundant** and results in **unnecessary bytecode and gas usage** during contract deployment.

Impact:

- Slight increase in contract size and deployment cost
- Reduced code clarity due to redundant assignments
- Missed opportunity for gas optimization

Recommendation

Avoid explicitly initializing variables to their default values. Rely on Solidity's automatic default initialization to reduce bytecode size and deployment gas costs.

Alleviation

--

References

https://docs.soliditylang.org/en/latest/style-guide.html#initial-values

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

PDM-01: Use of Post-Increment `i++` Instead of Pre-Increment `++i` in Loops

Vulnerabilities	Severity	Category	Status
Use of Post-Increment `i++` Instead of Pre-Increment `++i` in Loops	Informational	Gas Optimization	Acknowledge

Finding Code

VCTPresaleV2.sol:

```
145:         for (uint256 i = 0; i < STAGE_COUNT; i++) {
238:         for (uint256 i = start; i < end; i++) {
319:         for (uint256 i = 0; i < 4; i++) {
364:         currentStage++;
```

VCTPresaleV2.sol:

```
94:         for (uint256 i = 0; i < reserves.length; i++) {
```

Description

The contract uses the post-increment (i++) or post-decrement (i--) operator in a for-loop. While this is syntactically valid, using pre-increment (++i) or pre-decrement (--i) is marginally more gas-efficient in Solidity.

This is because post-increment creates a temporary copy of the variable before incrementing, whereas pre-increment directly increments the variable, avoiding unnecessary stack operations. Although the gas difference is small, this optimization is considered a best practice, especially within loops that may execute multiple iterations.

Recommendation

Replace all instances of `i++` / `i--` in loop expressions with `++i` / `--i` for slight gas savings and improved efficiency, especially when the return value of the increment/decrement expression is not used.

Alleviation

--

References

--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

IRU-01: Inefficient `require()` Usage with && Conditions

Vulnerabilities	Severity	Category	Status
Inefficient `require()` Usage with && Conditions	Informational	Gas Optimization	Acknowledge

Finding Code

File: `VCTPresaleV2.sol`

```
344:         require(block.timestamp >= s.startTime && block.timestamp <= s.endTime, "Presale: stage
not active");
```

Description

The contract uses require() statements that contain **multiple conditions joined by the logical AND operator (&&)**. While functionally correct, this pattern is slightly less gas-efficient compared to splitting the conditions into **separate require() statements**.

This is because Solidity will evaluate **all conditions** in the combined require() before generating a revert string, while multiple require() statements can short-circuit earlier and provide **clearer error reporting**.

Recommendation

Refactor require() statements that combine conditions with && into **separate require() calls**. This improves **gas efficiency, clarity, and debugging experience**.

Alleviation

--

References

--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

CAL-01: Uncached Array Length Access in Loop

Vulnerabilities	Severity	Category	Status
Uncached Array Length Access in Loop	Informational	Gas Optimization	Acknowledge

Finding Code

File: `VCTToken.sol`

```
94:         for (uint256 i = 0; i < reserves.length; i++) {
```

Description

The contract accesses an array's `.length` property directly within the condition of a for loop. While functionally correct, this pattern may lead to unnecessary gas consumption, especially if the array resides in storage rather than memory.

In Solidity, accessing a storage array's `.length` property is not free — each access incurs a read operation, which consumes gas. If the length is accessed in every iteration of the loop, the cost accumulates.

Recommendation

Cache the array's length in a local variable before entering the loop. This reduces repeated storage reads and improves gas efficiency without affecting functionality.

Alleviation

--

References

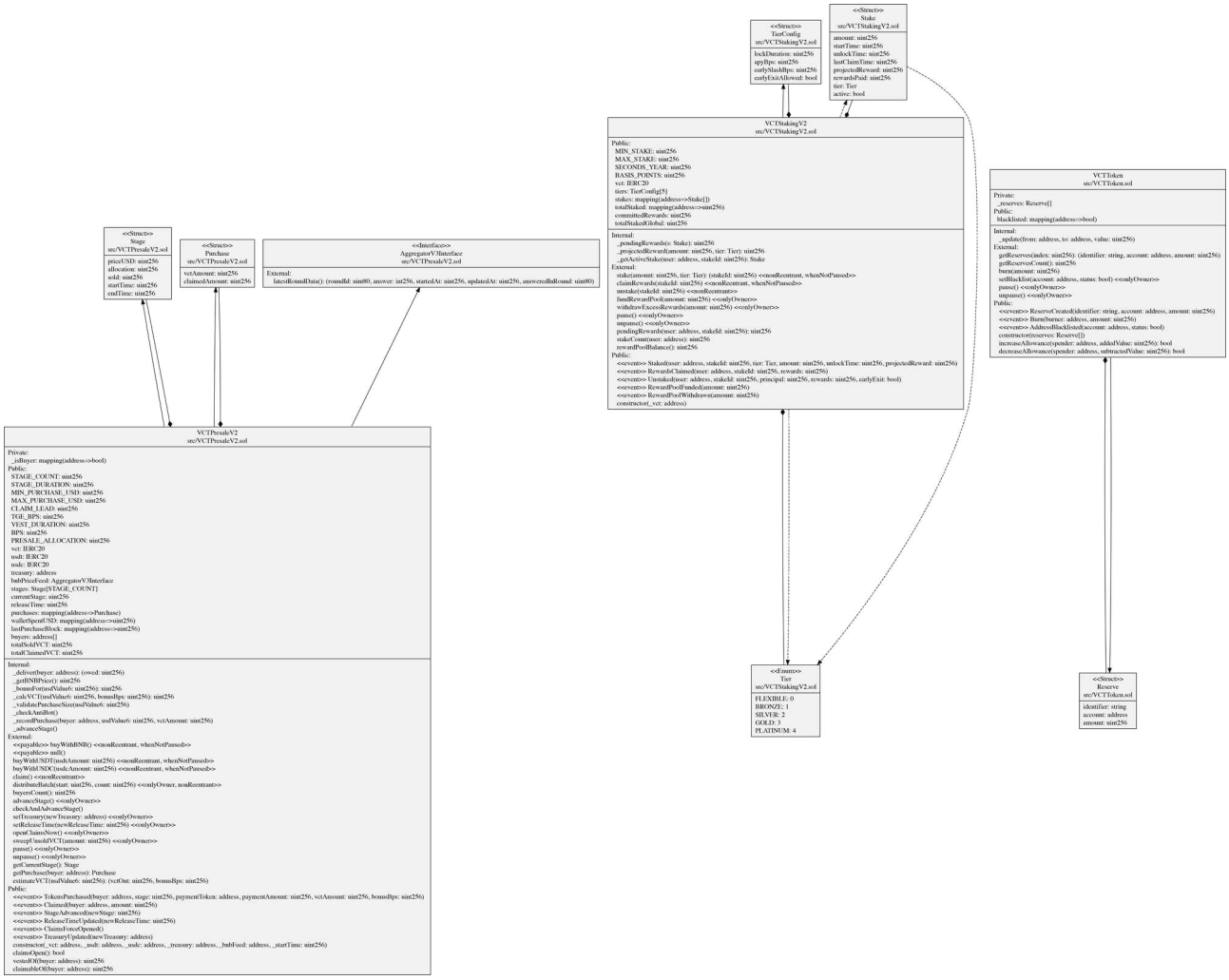
--

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

A Appendix

A.1 UML Class Diagram



© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

A.2 Contracts Description Table

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
AggregatorV3Interface	Interface			
└	latestRoundData	External		NO
VCTPresaleV2	Implementation	ReentrancyGuard, Ownable2Step, Pausable		
└		Public		Ownable
└	buyWithUSDT	External		nonReentrant whenNotPaused
└	buyWithUSDC	External		nonReentrant whenNotPaused
└	buyWithBNB	External		nonReentrant whenNotPaused
└	claimsOpen	Public		NO
└	vestedOf	Public		NO
└	claimableOf	Public		NO
└	claim	External		nonReentrant
└	distributeBatch	External		onlyOwner nonReentrant
└	_deliver	Internal		
└	buyersCount	External		NO
└	advanceStage	External		onlyOwner
└	checkAndAdvanceStage	External		NO
└	setTreasury	External		onlyOwner
└	setReleaseTime	External		onlyOwner
└	openClaimsNow	External		onlyOwner
└	sweepUnsoldVCT	External		onlyOwner

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Contract	Type	Bases		
└	pause	External		onlyOwner
└	unpause	External		onlyOwner
└	getCurrentStage	External		NO
└	getPurchase	External		NO
└	estimateVCT	External		NO
└	_getBNBPrice	Internal		
└	_bonusFor	Internal		
└	_calcVCT	Internal		
└	_validatePurchaseSize	Internal		
└	_checkAntiBot	Internal		
└	_recordPurchase	Internal		
└	_advanceStage	Internal		
└		External		NO
VCTToken	Implementation	ERC20, ERC20Pausable, Ownable2Step		
└		Public		ERC20 Ownable
└	getReserves	External		NO
└	getReservesCount	External		NO
└	burn	External		NO
└	increaseAllowance	Public		NO
└	decreaseAllowance	Public		NO
└	setBlacklist	External		onlyOwner
└	pause	External		onlyOwner
└	unpause	External		onlyOwner
└	_update	Internal		
VCTStakingV2	Implementation	ReentrancyGuard, Ownable2Step, Pausable		

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Contract	Type	Bases		
└		Public		Ownable
└	stake	External		nonReentrant whenNotPaused
└	claimRewards	External		nonReentrant whenNotPaused
└	unstake	External		nonReentrant
└	fundRewardPool	External		onlyOwner
└	withdrawExcessRewards	External		onlyOwner
└	pause	External		onlyOwner
└	unpause	External		onlyOwner
└	pendingRewards	External		NO
└	stakeCount	External		NO
└	rewardPoolBalance	External		NO
└	_pendingRewards	Internal		
└	_projectedReward	Internal		
└	_getActiveStake	Internal		

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

A.3 Components / Dependencies

Components

****Contracts	****Libraries	****Interfaces	****Abstract
3	0	1	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

****Public	****Payable
40	2

External	Internal	Private	Pure	View
35	45	0	1	19

StateVariables

Total	****Public
36	34

Capabilities

Solidity Versions observed	** Experimental Features**	** Can Receive Funds**	** Uses Assembly**	** Has Destroyable Contracts**
^0.8.20 0.8.26		yes	***	***

** Transfers ETH**	**** Low-Level Calls	** DelegateCall**	** Uses Hash Functions**	** ECRecover**	** New/Create/Create2**
***	***	***	***	***	***

** TryCatch**	Σ Unchecked
	yes

Dependencies / External Imports

Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable2Step.sol	3

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

Dependency / Import Path	Count
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	2
@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	2
@openzeppelin/contracts/utils/Pausable.sol	2
@openzeppelin/contracts/utils/ReentrancyGuard.sol	2

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

A.4 Source Units in Scope

Source Units Analyzed: **3**
Source Units in Scope: **3** (**100%**)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	src/VCTPresaleV2.sol	1	1	398	363	259	110	212	
	src/VCTToken.sol	1	***	185	175	70	80	64	**Σ**
	src/VCTStakingV2.sol	1	***	224	224	153	47	113	
	Totals	**3**	**1**	**807**	**762**	**482**	**237**	**389**	****Σ**

Legend: [ ]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC

About SCRL

SCRL is a dynamic cybersecurity team that provides security assessments, penetration testing, and incident investigations. We utilize specialized tools, expertise, and frameworks to enhance security measures. Additionally, we develop robust internal tools tailored to our organization's needs.
SCRL is committed to driving security for the blockchain industry and businesses alike.

Follow Us On:

Platform	Link
Website	https://scrl.io/
Twitter	https://twitter.com/scrl_io
Telegram	https://t.me/scrl_io
Medium	https://scrl.medium.com/

© 2026 SCRL. All rights reserved. | https://scrl.io/project/valletoken
This security assessment does not provide investment advice

Document ID: EC1E2A87CCF1BE980BD70FBC260083AC