

ABRACADABRA MONEY · SECURITY REVIEW

Abracadabra Money

PurrSwap ve33.

A security review of PurrSwap ve33.

CLIENT

Abracadabra
Money

PROJECT

PurrSwap ve33

AUDITORS

Nicholas Chew,
0xCiphky .,
Zdravko Hristov

AUDIT WINDOW

May 19 – June 4,
2025

FINAL REPORT

June 4, 2025

§ 00

Contents.

97 pages: overview, confidence ranking, scope, 74 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
§ 04	Findings	P. 06
	Main Review · 74 findings	P. 09
§ 05	Disclaimer	P. 96
§ 06	About Guardian	P. 97

§ 01

Overview.

Guardian reviewed PurrSwap ve33 across a single review round. The final report includes 5 Critical, 10 High, 20 Medium, 39 Low findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/GuardianOrg/purrswap-contractspurrswap-ve33-team1
CLIENT	METHODOLOGY
Abracadabra Money	Smart contract · Manual · Main Review
PROJECT	MAIN REVIEW COMMIT
PurrSwap ve33	d00d434216bc
PREPARED BY	
Nicholas Chew, 0xCiphky ., Zdravko Hristov	
AUDIT WINDOW	
May 19 – June 4, 2025	
FINAL REPORT	
June 4, 2025	

1.1 Confidence Ranking

1 / 5

Very Low Confidence

Guardian assigns a Confidence Ranking of 1 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.



1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
■ Critical	5	5	0	0	0
■ High	10	10	0	0	0
■ Medium	20	20	0	0	0
■ Low	39	39	0	0	0
■ Informational	0	0	0	0	0
All severities	74	74	0	0	0

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence ASSIGNED	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed PurrSwap ve33 in

```
https://github.com/Guardian0rg/purrrswap-contractspurrrswap-ve33-team1.
```

IN SCOPE

Deployment Scripts

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH
Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM
Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW
Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH
Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM
Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW
Unlikely to ever occur in production.

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
C-01	Broken Vote System As Gauge Weights Do Not Decay	Logical Error	CRITICAL	PENDING	p. 9
C-02	Poke can be blocked	Logical Error	CRITICAL	PENDING	p. 11
C-03	checkpointTotalSupply Caches Supply Too Early	Logical Error	CRITICAL	PENDING	p. 12
C-04	Anyone can block Gauge withdrawals	Logical Error	CRITICAL	PENDING	p. 13
C-05	Weekly emissions can be blocked	Unexpected Behavior	CRITICAL	PENDING	p. 14
H-01	Rewards Cannot Be Claimed After veNFT Expires	Logical Error	HIGH	PENDING	p. 15
H-02	Lock Time Can Be Reduced via Reentrancy	Logical Error	HIGH	PENDING	p. 17
H-03	Votes remain active in perpetuity with no way to reset	—	HIGH	PENDING	p. 19
H-04	Claim of growth emissions can be grieved	DoS	HIGH	PENDING	p. 20
H-05	Stuck Rewards in VotingEscrowDistributor	Logical Error	HIGH	PENDING	p. 22
H-06	Unsafe Casting Leads To Underflow	Math	HIGH	PENDING	p. 23
H-07	Incorrect Calculation Of Team Emissions	Logical Error	HIGH	PENDING	p. 25
H-08	Staked LP Fees Cannot Be Claimed	Logical Error	HIGH	PENDING	p. 26
H-09	Low-Decimal Token Rewards Can Be Grieved	Rounding	HIGH	PENDING	p. 27
H-10	Checkpoint overload caused by getReward	DoS	HIGH	PENDING	p. 29
M-01	Malicious user can fill rewards array in gauge and bribes differently	—	MEDIUM	PENDING	p. 30
M-02	Incorrect Voting Power From balanceOfNFT	Logical Error	MEDIUM	PENDING	p. 31
M-03	Improper Supply Update on Merge	Logical Error	MEDIUM	PENDING	p. 32
M-04	Rewards can be grieved for lower decimal tokens	Rewards	MEDIUM	PENDING	p. 33
M-05	veForAt() returning enormous value	Unexpected Behavior	MEDIUM	PENDING	p. 37
M-06	updatePeriod Reverts When No Votes Are Cast	DoS	MEDIUM	PENDING	p. 38

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
M-07	Initial veDistributor rewards not claimable	Rewards	MEDIUM	PENDING	p. 39
M-08	Merge Of Expired Locks Breaks Claims	Logical Error	MEDIUM	PENDING	p. 40
M-09	Incorrect initialLastPoint Copy Breaks Snapshot	Logical Error	MEDIUM	PENDING	p. 41
M-10	Loss Of Unclaimed Rewards And Bribes	Logical Error	MEDIUM	PENDING	p. 43
M-11	Griefing Via Bribe Rewards Token Length	DoS	MEDIUM	PENDING	p. 44
M-12	killGauge May Reset Unsent Rewards	Logical Error	MEDIUM	PENDING	p. 45
M-13	Lost emissions due to killed gauges	Rewards	MEDIUM	PENDING	p. 46
M-14	Killed gauge may accrue rewards	Unexpected Behavior	MEDIUM	PENDING	p. 47
M-15	Enabling autocompounding breaks the system	Configuration	MEDIUM	PENDING	p. 48
M-16	Ineffective Burn of PSPS Tokens in burn	Logical Error	MEDIUM	PENDING	p. 49
M-17	Precision Loss Leads To Irrecoverable Rewards	Rounding	MEDIUM	PENDING	p. 50
M-18	Restricted poke call frequency	Unexpected Behavior	MEDIUM	PENDING	p. 51
M-19	Incorrect Epoch Accounting for Rewards	Logical Error	MEDIUM	PENDING	p. 52
M-20	Wrong reward validation in Bribe	Validation	MEDIUM	PENDING	p. 53
L-01	Duplicate read from storage	—	LOW	PENDING	p. 54
L-02	Malicious Admin can frontrun whitelist()	—	LOW	PENDING	p. 55
L-03	Tokens that are delisted will remain active in existing gauges	—	LOW	PENDING	p. 56
L-04	Permissionless pool and gauge deployments can DoS updates and distributions	—	LOW	PENDING	p. 57
L-05	Inconsistent Flash-Loan Block Protection Across balanceOfNFT Variants	Logical Error	LOW	PENDING	p. 58
L-06	Incorrect view function listingFee()	—	LOW	PENDING	p. 59
L-07	Two files with the same name	Best Practices	LOW	PENDING	p. 60
L-08	VotingEscrow is not ERC721 compliant	Unexpected Behavior	LOW	PENDING	p. 61
L-09	Duplicate function distro() can be removed	—	LOW	PENDING	p. 62
L-10	User can create false emitWithdraw emissions	—	LOW	PENDING	p. 63
L-11	Minter initializing can be blocked	Informational	LOW	PENDING	p. 64
L-12	Minter doesn't follow the CEI pattern	Best Practices	LOW	PENDING	p. 65
L-13	Inaccurate Minter MAX_LOCK	Math	LOW	PENDING	p. 66
L-14	listingFee Reverts If Fee Not Set	Unexpected Behavior	LOW	PENDING	p. 67

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
L-15	initialize Overrides Intended Emissions Delay	Logical Error	LOW	PENDING	p. 68
L-16	Rewards claim underflows for invalid tokens	Unexpected Behavior	LOW	PENDING	p. 69
L-17	Rewards can be recovered from MultiRewards	Rewards	LOW	PENDING	p. 70
L-18	Points can leave not whitelisted wallets	Validation	LOW	PENDING	p. 72
L-19	Part of MultiRewards may not be claimed	Rewards	LOW	PENDING	p. 73
L-20	No rewards length cap in MultiRewards	Rewards	LOW	PENDING	p. 74
L-21	MultiRewards tokens can be blacklisted	Rewards	LOW	PENDING	p. 75
L-22	No Removal Function for Inactive Reward Tokens	Best Practices	LOW	PENDING	p. 77
L-23	Operator Receives Funds Instead of veNFT Owner	Unexpected Behavior	LOW	PENDING	p. 78
L-24	ownershipChange Can Be Bypassed With merge	Logical Error	LOW	PENDING	p. 79
L-25	Ambiguous event emission	Events	LOW	PENDING	p. 80
L-26	Precision Loss Leaves Reward Dust	Rounding	LOW	PENDING	p. 81
L-27	Inconsistent veNFT Access Control in deposit	Access Control	LOW	PENDING	p. 82
L-28	Duplicate import statement	—	LOW	PENDING	p. 83
L-29	First Epoch Emissions Over-Boosted	Informational	LOW	PENDING	p. 84
L-30	balanceOfAtNFT() is not fully accurate	Warning	LOW	PENDING	p. 85
L-31	Wrong owner for Minter deployment	Configuration	LOW	PENDING	p. 86
L-32	Lack of artProxy	Configuration	LOW	PENDING	p. 88
L-33	Only 2 tokens whitelisted	Informational	LOW	PENDING	p. 89
L-34	Incorrect Mapping and Detach Logic In Withdraw	DoS	LOW	PENDING	p. 90
L-35	Duplicate Reward Tokens Possible via Frontrun	Frontrunning	LOW	PENDING	p. 91
L-36	Missing ERC20 Recover Functions in Contracts	Informational	LOW	PENDING	p. 92
L-37	Wrong maxRuns usage	Best Practices	LOW	PENDING	p. 93
L-38	Some voter functions can be DOSed	Informational	LOW	PENDING	p. 94
L-39	Bribe.left() is not necessary	Informational	LOW	PENDING	p. 95

CRITICAL C-01

PENDING

Broken Vote System As Gauge Weights Do Not Decay

CATEGORY Logical Error

LOCATION Voter.sol

REVIEW Main Review

DESCRIPTION

When a user votes for a gauge, the contract increases `weights[_gauge]` by their current voting power. However, this gauge weight does not decay over time, even though the user's underlying voting power does. If the user does not vote again, the gauge retains an inflated weight indefinitely.

This results in a broken system of weekly emissions, as gauges backed by inactive voters continue receiving rewards disproportionate to their actual voting power, while active voters are penalized.

While the `poke` function exists to address this issue by updating stale weights, it has notable limitations:

- Only the veNFT owner can call `poke`. A malicious user can simply avoid calling it, indefinitely locking in inflated voting weight.
- If the user's voting power has fully decayed, `poke` reverts, making it impossible to clear the stale vote.

RECOMMENDATION

1. Consider implementing epoch-based voting. Require users to re-cast their vote each epoch (e.g., weekly). This ensures weights are only valid within the epoch they were cast.

2. Reframe `poke` as a convenience function that allows users to re-cast their vote for the current epoch, rather than a mechanism to correct abuse. The current access control (owner-only) can remain, but the misleading NatSpec comment:

```
/// @notice To be called on voters abusing their voting power  
should be removed.
```

PROOF OF CONCEPT <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/pull/2/files>

C-01 Broken Vote System As Gauge Weights Do Not Decay

CONTINUED

— RESOLUTION

Status: Pending.

CRITICAL C-02

PENDING

Poke can be blocked

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/bb94a935ec97ff65b8c91a38cb98a864f637e535/contracts/Voter.sol#L509
REVIEW	Main Review

— DESCRIPTION

The poke function in the Voter contract is intended to update a user's voting power, especially in cases where users may be abusing the system by allowing outdated voting weights to persist. For instance, a user who votes once with a long-term lock (e.g., 4 years) retains their full initial voting weight indefinitely, unless they later call vote, poke, or reset.

A separate issue (reported) restricts the ability to call poke to the token owner. However, even if this limitation is addressed (e.g., by allowing the protocol to call poke), users can still block the function under certain conditions.

The issue lies in the internal `_vote` function. If a user's voting power has fully decayed—such as after their lock has expired—the resulting gauge weights are all zero. Since `_vote` requires gauge weights to be greater than zero, the function reverts, preventing the update from proceeding via poke.

A more impactful attack involves the user assigning a gauge weight just above zero for one of the gauges. As their voting power decays, the computed gauge weight eventually rounds down to zero. If poke is called at that point, it reverts due to the zero-weight gauge, effectively blocking the function. This allows the user to preserve outdated voting power and continue receiving disproportionate voting influence and rewards.

— RECOMMENDATION

Update the poke logic to avoid reverting when a calculated gauge weight is zero—for example, by skipping such gauges rather than failing the entire transaction.

PROOF OF CONCEPT <https://gist.github.com/ZdravkoHr/f9e3ce8fcf3bada5262d1edf9c364eb0>

— RESOLUTION

Status: Pending.

CRITICAL C-03

PENDING

checkpointTotalSupply Caches Supply Too Early

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrowDistributor.sol#L236
REVIEW	Main Review

DESCRIPTION

The `checkpointTotalSupply` function caches the total voting points for a given week by calling `VotingEscrow.pointHistory` and storing the value for use in reward calculations.

However, it caches the value at the start of the epoch, not the end. Since users can lock or unlock tokens during the week, this cached supply can quickly become stale — opening the door for manipulation.

Example attack:

- Alice calls `checkpointTotalSupply`, caching a supply of `1e18`.
- She then locks `2e18` worth of tokens, increasing the total supply to `3e18`.
- When claiming rewards, her share is calculated as:

`2e18 * rewards / 1e18` → she receives double the intended allocation.

Because the system divides user balances by an outdated (lower) `totalSupply`, it inflates her share at the expense of others.

RECOMMENDATION

Update the loop condition to break when `t == roundedTimestamp`:

```
``plain text
for (uint256 i = 0; i < MAX_PAST_PERIODS; i++) {
  if (t >= roundedTimestamp) {
    break;
  }
}
```

RESOLUTION

Status: Pending.

CRITICAL **C-04**

PENDING

Anyone can block Gauge withdrawals

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Gauge.sol#L171
REVIEW	Main Review

— DESCRIPTION

Withdrawals in Gauge.sol can be blocked by writing numerous supply checkpoints using the `getReward()` function. The issue lies in the fact that `getReward()` updates the reward checkpointing of the specific token passed into it, not all of the reward tokens. This function writes a supply checkpoint at the end execution. Meanwhile, all deposit and withdraw functions update all reward tokens' checkpoints. Updating a reward token checkpoint involves looping through each new supply checkpoint and writing to storage. Therefore, by creating numerous supply checkpoints while not updating the reward checkpoints of the other tokens, all user withdrawals will revert due to OOG error when the function attempts to update all of the reward checkpoints.

— RECOMMENDATION

Either call `_updateRewardForAllTokens();` at the start of the `getReward()` function or remove the supply checkpointing logic at the end of the `getReward()` function. It is not necessary as supply does not change.

PROOF OF CONCEPT <https://gist.github.com/fatherGoose1/0d6623ac6450fc7c24d330d7d574a128>

— RESOLUTION

Status: Pending.

CRITICAL C-05

PENDING

Weekly emissions can be blocked

CATEGORY Unexpected Behavior

LOCATION Minter.sol#L232

REVIEW Main Review

DESCRIPTION

Each week `Minter.updatePeriod()` is called to distribute weekly emissions, part of which are the `growth` emissions, calculated in `_calculateGrowth()`. That function fetches the total `bias` for 1 second before `activePeriod` by calling `totalSupplyAtT()`

```
``plain text
```

```
ve.totalSupplyAtT(activePeriod - 1)
```

```
`
```

The problem is that `totalSupplyAtT()` will revert if it's called for a past timestamp and there is a checkpoint created at any point in time after that timestamp. This is because it subtracts the checkpoint timestamp from the desired timestamp, while both of them are `uint256`, resulting in a revert with underflow.

```
`plain text
```

```
tI - lastPoint.ts
```

```
`
```

Anyone can call `ve.checkpoint()` and record a checkpoint for a given timestamp. Therefore, if a checkpoint is created before `updatePeriod()` is called, it will be impossible to distribute the rewards for that whole period. The next week the Minter will try distributing the same rewards, but this will affect PSPS' lockers and bribes. Of course, this attack can be repeated again, keeping the emissions blocked.

RECOMMENDATION

You can use the utility function `_findTimestampEpoch()` from the other files to find the epoch which is before or at `block.timestamp - 1` (not `activePeriod - 1`, since `_calculateGrowth()` fetches the current `totalSupply()` but uses locked value from the past). After that calculate the voting power by adjusting the `bias` using the `slope`.

RESOLUTION

Status: Pending.

HIGH **H-01**

PENDING

Rewards Cannot Be Claimed After veNFT Expires

CATEGORY	Logical Error
LOCATION	VotingEscrowDistributor.sol: 79, 105
REVIEW	Main Review

— DESCRIPTION

In the current design, rewards for veNFT holders are distributed through a `claim` function that forces the newly minted tokens to be deposited back into the lock. However, if a veNFT has passed its lock end time, any attempt to call `IVotingEscrow(ve).depositFor(_tokenId, amount)` will revert due to the following requirement inside `depositFor`:

```
`solidity
require(oldLocked.end > block.timestamp, "cannot add to expired lock, withdraw");
`
```

Once the veNFT is expired, users can no longer deposit newly claimed rewards into it. This reverts the entire reward claim, causing those tokens to remain stuck in the `VotingEscrowDistributor` contract. As a result, users lose access to their rewards if they attempt to claim after the lock expires. This issue affects the `VotingEscrowDistributor.claim` and `VotingEscrowDistributor.claimMany` functions.

— RECOMMENDATION

H-01 Rewards Cannot Be Claimed After veNFT Expires

CONTINUED

Allow expired veNFTs to claim and receive their rewards directly (rather than forcing `depositFor`). A possible fix is to check whether the lock is expired. If it is, skip the `depositFor` call and transfer the rewards to the original veNFT owner instead:

```
``solidity
if (oldLocked.end > block.timestamp) {
  IVotingEscrow(ve).depositFor(_tokenId, amount);
} else {
  // Lock is expired
  // Transfer rewards directly to veNFT owner
  address nftOwner = IVotingEscrow(ve).ownerOf(_tokenId);
  token.safeTransfer(nftOwner, amount);
}
``
```

This preserves the intended lock mechanism for active veNFTs while letting expired locks still claim any accrued tokens.

— RESOLUTION

Status: Pending.

HIGH **H-02**

PENDING

Lock Time Can Be Reduced via Reentrancy

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/VotingEscrow.sol#L272
REVIEW	Main Review

DESCRIPTION

The `createLockFor` and `createLock` functions in the `VotingEscrow` contract allow users to deposit tokens that are locked for a specified duration. These tokens can then be claimed by the designated receiver once the lock period has expired. For example, the `Minter` contract's `initialize` function uses `createLockFor` to create maximum-duration locks (four years) for initial participants.

The issue is that a user can exploit the `merge` function to significantly reduce this lock period. Here's how:

- User A is allocated 1,000 tokens via the `Minter` contract's `initialize` function, which are locked for four years and assigned to token ID 1.
- User A then calls `createLockFor`, locking 100 tokens for just one week. This creates a new lock under token ID 2.
- During this call, inside the `_mint` function, User A uses the `onERC721Received` callback to reenter the contract and call `merge`, combining token ID 1 (4-year lock) into token ID 2 (1-week lock). At this point, token ID 2 holds 1,000 tokens and retains the longer lock time (4 years).
- Once the merge completes, the original `createLockFor` call resumes. It continues to `_depositFor`, which adds the new 100-token deposit to token ID 2 and updates the lock time to the current call's duration—just one week.

As a result, token ID 2 now holds 1,100 tokens with a one-week lock, effectively bypassing the intended four-year restriction.

This can also be abused to maximize voting power and rewards by first setting the lock to the maximum duration, then using this exploit to unlock early—gaining the benefits of a long-term lock without honoring its full term.

RECOMMENDATION

Apply the `nonReentrant` modifier to the `merge` function to prevent it from being called via reentrancy during functions that include external calls.

PROOF OF CONCEPT https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/2d4fe7b727c2b0adaf19413c627e4c730e20b939/test/Guardian/reentrancy_poc.t.sol#L186

H-02 Lock Time Can Be Reduced via Reentrancy

CONTINUED

— RESOLUTION

Status: Pending.

HIGH **H-03**

PENDING

Votes remain active in perpetuity with no way to reset

LOCATION <https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L155-L175>

REVIEW Main Review

DESCRIPTION

After a user votes, their vote power will be applied to gauges forever without the ability for another actor to reset the weights. After the user's VE position expires, their vote power will remain attached to the gauge until the user intentionally calls `Voter.reset()`

Both `Voter.reset()` and `Voter.poke()` can only be called by the VE token owner or approved party. Therefore, there is no way to reset the token's vote weight to 0 after expiry. This results in the ability to vote for gauges without having a true lock. The user can reap the benefits of voting while maintaining the ability to withdraw whenever desired.

RECOMMENDATION

Remove the clause `require(IVotingEscrow(ve).isApprovedOrOwner(msg.sender, _tokenId), ErrUnauthorized());` from `Voter.poke()` to allow anyone the ability to reset vote weights.

PROOF OF CONCEPT <https://gist.github.com/fatherGoose1/4a21bd1329e3b338805632e430613ded>

RESOLUTION

Status: Pending.

HIGH **H-04**

PENDING

Claim of growth emissions can be grieved

CATEGORY	DoS
LOCATION	VotingEscrowDistributor.sol
REVIEW	Main Review

— DESCRIPTION

There is a `primary issue` which allows a malicious user to block claiming rewards of any other user and a `secondary issue` which makes it easier to keep the blocked state.

The `primary issue` is that `VotingEscrow.depositFor()` is permissionless. Any party can call it for a `tokenId` and write a checkpoint for it. This means that a malicious user can spam `depositFor` in every block to create a lot of locks for any user. In addition, the block time on Hyperliquid is ~ 0.2 seconds, which means they can technically create 5 checkpoints per second. When the fair user tries to claim their rewards, the for loop in `_claimable()` will try to find the closest checkpoint before the `weekCursor`. Once it finds it, it will move the cursor and continue. However, the loop is bounded by `MAX_CLAIM_ITERATIONS = 50`.

Because the malicious user created a lot of checkpoints, the function will end without rewarding the user or moving the cursor. The `userEpoch` will be increased with `50`. Then the user may call `claim()` a lot of times until they offset the malicious checkpoints. And here comes the secondary issue.

`Secondary issue` - when the `weekCursor` is beyond the `lastTokenTime`, the `userEpoch` is being returned from `_claimable`. The value is then subtracted by 1 and saved in `userEpochOf`. This can be leveraged by anyone to spam `claim` calls and reset the `userEpochOf` for any user back to 0. Because of this the loop will have to do all the iterations again on the next claim.

— RECOMMENDATION

1. Make `depositFor()` callable only by the authorized accounts for the `tokenId` and the `VotingEscrowDistributor` account. This will fix the `primary issue`.
2. To fixed the `secondary issue`, modify this [if statement](/1f68bda5828c8038b4efc5b296193bda?pvs=25) in the following way

```
```diff
- if (weekCursor >= lastTokenTime) {
+ if (weekCursor >= _lastTokenTime) {
- return (weekCursor, maxUserEpoch, userEpoch, 0);
+ return (weekCursor, maxUserEpoch, userEpoch + 1, 0);
```

**H-04** Claim of growth emissions can be grieved

CONTINUED

}

``

By replacing `lastTokenTime` (not rounded to weeks) with `_lastTokenTime` (rounded to weeks), the code will stop earlier if a claim cannot happen and the for loop won't be entered. Then the `1` added to `userEpoch` will cancel out with the `1` subtracted in `_claim()` and the `userEpochOf` will not change.

**PROOF OF CONCEPT** <https://gist.github.com/ZdravkoHr/0b9621269f4c47be9ba7fefb71efdb82>

**— RESOLUTION**

Status: Pending.

HIGH **H-05**

PENDING

## Stuck Rewards in VotingEscrowDistributor

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/VotingEscrowDistributor.sol#L69">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/VotingEscrowDistributor.sol#L69</a>
REVIEW	Main Review

### — DESCRIPTION

The claim function in the VotingEscrowDistributor contract attempts to redeposit any claimed rewards for a token into the VotingEscrow contract by calling the depositFor function. However, if the token's lock has expired, this deposit will revert, as it is not possible to deposit into an expired lock. Additionally, the lock duration cannot be extended after expiration, since increasing the lock time is only permitted while the lock is still active. As a result, the user's claim for that token becomes permanently stuck in the VotingEscrowDistributor contract.

### — RECOMMENDATION

Modify the claim function to check for expired locks and, if expired, transfer rewards directly to the token owner instead of attempting a deposit.

### — RESOLUTION

Status: Pending.

HIGH **H-06**

PENDING

# Unsafe Casting Leads To Underflow

CATEGORY Math

LOCATION <https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrowDistributor.sol#L318>, <https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrowDistributor.sol#L247>

REVIEW Main Review

## DESCRIPTION

The `_claimable` and `_checkpointTotalSupply` functions in `VotingEscrowDistributor` calculate user rewards and total `ve` token supply over time. Both perform decay calculations using:

```
```solidity
// _claimable()
uint256 balanceOf = Math.max(uint256(int256(oldUserPoint.bias - dt * oldUserPoint.slope)), 0);
```

```
// _checkpointTotalSupply()
veSupply[t] = Math.max(uint256(int256(pt.bias - pt.slope * dt)), 0);
``
```

However, if `dt * slope > bias`, the subtraction results in a negative value. Casting this to `uint256` causes an underflow, turning it into a large positive number. `Math.max` then returns this inflated value.

As a result, for `_claimable` the calculated `balanceOf` becomes an extremely large number, causing the claim transaction to revert as the contract does not have sufficient balance. Similarly, when `veSupply` underflows, it is set to a very large number. Since user rewards are distributed proportionally based on `userBalance / veSupply`, the result effectively rounds down to zero. This causes all users to receive no rewards for that week, regardless of their actual stake.

RECOMMENDATION

Do a `maxInt` comparison first then convert to `uint128`:

```
```solidity
// _claimable()
uint256 balanceOf = uint128(int256(Math.maxInt((oldUserPoint.bias - dt * oldUserPoint.slope), 0)));
```

**H-06** Unsafe Casting Leads To Underflow

CONTINUED

```
// _checkpointTotalSupply()
veSupply[t] = uint256(int256(Math.maxInt((pt.bias - pt.slope * dt), 0)));
...
```

**— RESOLUTION**

Status: Pending.

HIGH **H-07**

PENDING

# Incorrect Calculation Of Team Emissions

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Minter.sol#L129">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Minter.sol#L129</a>
REVIEW	Main Review

## DESCRIPTION

The `updatePeriod` function calculates weekly emissions for the Distributor, Voter, and Treasury (team). For the first 65 epochs, emissions are stored in both a local variable `_emission` and the storage variable `weekly`.

However, starting from epoch 66, the `weekly` storage variable is no longer updated. Despite this, team emissions are still calculated using `weekly`:

```
uint256 _teamEmissions = (teamRateInBips * (_growth + _weekly)) / (BIPS - teamRateInBips);
```

This uses an outdated value and ignores the freshly computed `_emission`. Over time, this results in team emissions being over-calculated, compounding each week and leading to exponential inflation.

## RECOMMENDATION

Update the calculation to use the local `_emission` variable instead of the stale `weekly` value:

```
uint256 _teamEmissions = (teamRateInBips * (_growth + _emission)) / (BIPS - teamRateInBips);
```

## RESOLUTION

Status: Pending.

HIGH **H-08**

PENDING

# Staked LP Fees Cannot Be Claimed

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/MultiRewards.sol">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/MultiRewards.sol</a>
REVIEW	Main Review

## — DESCRIPTION

The MultiRewards contract allows users to stake PurrLP tokens to earn rewards and points. However, while these LP tokens remain staked, they continue to generate trading fees internally within the PurrLP pair contract.

Currently, these accrued fees are not claimable, as MultiRewards does not expose any function to call `claimFees` on the underlying PurrLP. As a result:

- Trading fees accumulate but remain unutilized.
- LPs miss out on a key source of yield, which may reduce the attractiveness of staking.

## — RECOMMENDATION

Implement a function in MultiRewards to allow the contract (or a designated role like owner or operator) to call the fee-claiming logic on the underlying PurrLP.

## — RESOLUTION

Status: Pending.

HIGH **H-09**

PENDING

# Low-Decimal Token Rewards Can Be Griefed

CATEGORY Rounding

LOCATION <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/blob/main/contracts/Gauge.sol#L455>

REVIEW Main Review

## DESCRIPTION

In the Gauge contract, if the reward token is a low-decimal token like USDC (6 decimals), a low-cost grieving attack becomes possible. A malicious user can repeatedly call the `getReward` function every few blocks, triggering the `_updateReward` function and effectively nullifying USDC rewards.

Attack scenario:

1. A reward distributor sends 10,000 USDC to the Gauge via `notifyRewardAmount`. The `rewardRate` is calculated as  $10\_000e6 / 1 \text{ weeks}$ , which is approximately `16,534`.

2. Every 60 seconds (roughly 30 blocks on Hyperliquid EVM, assuming 2-second block times), the attacker calls `getReward`, which updates the reward accounting.

3. The reward per token is calculated in `_calcRewardPerToken` as:

$(currentTime - lastUpdateTime) * rewardRate[token] * 1e18 / derivedSupply$ .

1. If the time delta is small and the `rewardRate` is low due to USDC's 6 decimal places, the formula rounds down to zero.

For example, assuming a `derivedSupply` of 1 million tokens:

$60 * 16534 * 1e18 / 1\_000\_000e18 = 0.99204$ , which truncates to zero.

1. As `rewardPerTokenStored` remains zero, honest users receive little or no rewards when they try to claim. Over a one-week period, calling `getReward` every 60 seconds would cost the attacker only:

$604800 / 60 * \$0.01 = \$100.80$ , assuming  $\$0.01$  per transaction on Hyperliquid.

This makes the attack highly accessible and economically viable, while causing significant loss of rewards for legitimate participants.

## RECOMMENDATION

Normalize the `rewardRate` to `1e18` regardless of the reward token's decimal precision. When claiming rewards, convert the earned rewards back to the original decimal precision.

**PROOF OF CONCEPT** <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/pull/4/files>

---

**H-09** Low-Decimal Token Rewards Can Be GriefedCONTINUED

---

---

**— RESOLUTION**

Status: Pending.

HIGH **H-10**

PENDING

# Checkpoint overload caused by getReward

CATEGORY	DoS
LOCATION	Gauge.sol
REVIEW	Main Review

## DESCRIPTION

Whenever `Gauge.getReward()` is called, all of the reward data for all of tokens the user wants to claim are updated and after that a total supply checkpoint is recorded. However, the user can pass an empty `tokens` array to avoid the rewards update. This will result only in new checkpoint. If the user does that every block, they can spam the system with a lot of checkpoints without changing `lastUpdateTime` for any of the rewards. This allows them to create a temporary DOS for `Gauge.deposit` and `Gauge.withdraw()` because they call `_updateRewardForAllTokens()` which goes through all of the checkpoints from the last update until the current one and updates their data. Especially on HyperEVM, the block gas limit for fast blocks is only 2M. In result users will not be able to withdraw or deposit until batch updates for these checkpoints are executed.

## RECOMMENDATION

Consider changing the update logic in `getReward` so it also calls `_updateRewardForAllTokens()`. This will ensure every time the function is called, all of the rewards are updated and therefore there are no checkpoints left between `lastUpdateTime` and the last checkpoint.

## RESOLUTION

Status: Pending.

MEDIUM **M-01**

PENDING

# Malicious user can fill rewards array in gauge and bribes differently

REVIEW

Main Review

## DESCRIPTION

The Bribe contract contain a function `notifyRewardAmount()` which attempts to add the specified token to the reward tokens array if it is not already included. The function checks that the specified token is whitelisted in the voter contract.

`Voter.sol` allows anyone to list whitelist a token if `permissionMode` is set to false. Therefore, a user can whitelist many tokens and populate the reward tokens in `Bribe.sol` with various tokens that are not `IPurrLP(stake).*BASE_TOKEN*()`; and `IPurrLP(stake).QUOTE*_TOKEN*()`; Subsequently, when `Gauge.claimFees()` is called, it will revert because the call `bribe.notifyRewardAmount()` will attempt to add the base and quote tokens to the rewards array that is already full.

## RECOMMENDATION

When creating the bribe contract, add the base and quote tokens of the pool to the rewards array to ensure that these tokens can be received as rewards.

## RESOLUTION

Status: Pending.

MEDIUM **M-02**

PENDING

# Incorrect Voting Power From balanceOfNFT

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L504">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L504</a>
REVIEW	Main Review

## DESCRIPTION

The `balanceOfNFT(uint256 _tokenId, uint256 _t)` function returns the voting power of a veNFT at timestamp `_t`, which is critical for snapshot-based voting.

However, the function incorrectly handles historical timestamps. Specifically, when `_t` is earlier than the user's last checkpoint, it erroneously increases the reported voting power. This allows users who staked after the snapshot to receive non-zero voting power, violating the intended logic that only tokens staked before the snapshot should count.

The root cause lies in this line:

```
lastPoint.bias -= lastPoint.slope * int128(int256(_t) - int256(lastPoint.ts));
```

If `_t < lastPoint.ts`, the subtraction becomes negative, and the result is effectively added to bias, artificially inflating the voting power.

Example

- User stakes at time `t = 10`
- Snapshot for voting is taken at time `t = 5`
- Voting power is calculated as:

```
bias - slope * (5 - 10) = bias + (5 * slope)
```

This gives the user positive voting power at `t = 5`, even though they had not staked yet. The earlier the snapshot, the higher the incorrect voting power becomes.

## RECOMMENDATION

Guard against timestamps earlier than the user's last checkpoint by returning zero voting power in such cases.

## RESOLUTION

Status: Pending.

MEDIUM **M-03**

PENDING

# Improper Supply Update on Merge

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/VotingEscrow.sol#L272">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/VotingEscrow.sol#L272</a>
REVIEW	Main Review

## — DESCRIPTION

The `merge` function combines two tokens into one by adding their amounts, using the later lock end time, and burning one of the token IDs. The problem is that after burning the source token, the function does **not** reduce the `supply` variable by its value.

It then proceeds to call `_depositFor`, which adds `value0` to the destination token and increases the `supply` by the same amount. Since no new tokens are being deposited—only merged—this results in an incorrect net increase in total supply.

Because this occurs on every merge call, the `supply` variable becomes increasingly inaccurate over time, diverging from the actual amount of tokens and breaking any logic that depends on a correct total supply.

## — RECOMMENDATION

Subtract `value0` from the `supply` variable during the merge function. Alternatively, modify `_depositFor` to skip increasing the `supply` variable when called as part of a merge.

## — RESOLUTION

Status: Pending.

MEDIUM **M-04**

PENDING

# Rewards can be grieved for lower decimal tokens

CATEGORY Rewards  
LOCATION MultiRewards.sol  
REVIEW Main Review

## DESCRIPTION

The `rewardRate` in `MultiRewards` is calculated as `amount / _rewardData[rewardToken].rewardsDuration`, where `amount` is in the token's decimals. For tokens with lower decimal precision (for example, 6), this value will be relatively small. The following calculation is then present in `rewardPerToken()`

```
``plain text
pendingRewardsPerToken = (timeElapsed * _rewardData[rewardToken].rewardRate * 1e18) /
totalSupply;
```

Whenever `timeElapsed * rewardRate * 1e18 < totalSupply`, `rewardPerToken()` will return the same value as before. This is a problem because whenever `_updateRewards()` is called, both `rewardPerTokenStored` and `lastUpdateTime` are updated. Since `rewardPerToken()` returns the old value and is assigned to `rewardPerTokenStored`, this means only `lastUpdateTime` will be effectively updated.

A malicious user can call `getRewards()` as frequently as possible to trigger execution of `_getRewards()`. Over time, this will result in no updates for `rewardPerTokenStored`, only `lastUpdateTime`. As a result, rewards are not accumulating and stakers won't be able to claim them.

## RECOMMENDATION

It's in notion

1. Add a `scalingFactor` field to the `Reward` struct

```
```diff
struct Reward {
uint256 rewardsDuration;
uint256 periodFinish;
uint256 rewardRate;
uint256 lastUpdateTime;
uint256 rewardPerTokenStored;
```

M-04 Rewards can be grieved for lower decimal tokens

CONTINUED

```

+ uint256 scalingFactor;
}

``|
1. Modify addReward() to accept how much decimals does the reward token have and calculate
the correct scaling factor
``diff
- function addReward(address rewardToken, uint256 _rewardsDuration) public onlyOwner {
+ function addReward(address rewardToken, uint256 _rewardsDuration, uint256 decimals) public onlyOwner
{
if (rewardToken == address(0)) {
revert ErrInvalidTokenAddress();
}
if (_rewardData[rewardToken].rewardsDuration != 0) {
revert ErrRewardAlreadyAdded();
}
if (_rewardsDuration == 0) {
revert ErrZeroDuration();
}

rewardTokens.push(rewardToken);
_rewardData[rewardToken].rewardsDuration = _rewardsDuration;
+ _rewardData[rewardToken].scalingFactor = _getScalingFactor(decimals);
}

``|
1. Create the _getScalingFactor() function. If you want to, you can make 18 a constant so it's
not a magic value.
``solidity
function _getScalingFactor(uint256 decimals) internal returns(uint256) {
if (decimals >= 18) {
return 1;
}

return 10 ** (18 - decimals);
}

``|
1. Adjust rewardRate by scalingFactor in notifyRewardAmount
``diff
function notifyRewardAmount(address rewardToken, uint256 amount) external
onlyOwnerOrRoles(ROLE_REWARD_DISTRIBUTOR) {

```

M-04 Rewards can be grieved for lower decimal tokens

CONTINUED

```

if (_rewardData[rewardToken].rewardsDuration == 0) {
    revert ErrInvalidTokenAddress();
}

_updateRewards(address(0));
rewardToken.safeTransferFrom(msg.sender, address(this), amount);
+ uint256 scalingFactor = _rewardData[rewardToken].scalingFactor;
+ amount *= scalingFactor;
// Take the remainder of the current rewards and add it to the amount for the next period
if (block.timestamp < _rewardData[rewardToken].periodFinish) {
    amount += (_rewardData[rewardToken].periodFinish - block.timestamp) *
        _rewardData[rewardToken].rewardRate;
}

_rewardData[rewardToken].rewardRate = amount / _rewardData[rewardToken].rewardsDuration;
_rewardData[rewardToken].lastUpdateTime = block.timestamp;
_rewardData[rewardToken].periodFinish = block.timestamp + _rewardData[rewardToken].rewardsDuration;
- emit LogRewardAdded(amount);
+ emit LogRewardAdded(amount / scalingFactor);
}

``
1. Undo the scaling in _getRewardsFor()
``diff
for (uint256 i; i < rewardTokens.length; ) {
    address rewardToken = rewardTokens[i];
    - uint256 reward = rewards[user][rewardToken];
    + uint256 reward = rewards[user][rewardToken] / _rewardData[rewardToken].scalingFactor;

    if (reward > 0) {
        rewards[user][rewardToken] = 0;
        rewardToken.safeTransfer(user, reward);
        emit LogRewardPaid(user, rewardToken, reward);
    }

    unchecked {
        ++i;
    }
}

```

M-04 Rewards can be grieved for lower decimal tokens

CONTINUED

```
``|
1. Finally, modify the getRewardForDuration() view function to account for the scaling factor
as well, so the result is accurate
``diff
function getRewardForDuration(address rewardToken) external view returns (uint256) {
- return _rewardData[rewardToken].rewardRate * _rewardData[rewardToken].rewardsDuration;
+ return _rewardData[rewardToken].rewardRate * _rewardData[rewardToken].rewardsDuration /
  _rewardData[rewardToken].scalingFactor;
}
...

```

PROOF OF CONCEPT <https://gist.github.com/ZdravkoHr/2a39948bdaed773441d0677b5f1c15e9>

— RESOLUTION

Status: Pending.

MEDIUM **M-05**

PENDING

veForAt () returning enormous value

CATEGORY	Unexpected Behavior
LOCATION	VotingEscrowDistributor.sol
REVIEW	Main Review

— DESCRIPTION

The `VotingEscrowDistributor.veForAt()` function returns the voting power of the NFT at a given point in time. If the `timestamp` which we are checking for is beyond the lock expiry, the calculation will result in a negative number. That's why the code tries to cap the returned value to 0. However, it turns the negative result into `uint256` and passes it after that to `Math.max()`. This casting result in an unreasonably large number, leading to extremely large values returned by `veForAt()`.

The same way of calculation is present in `_checkpointTotalSupply()` and `_claimable()`.

— RECOMMENDATION

Introduce a new override of the `Math.max()` function which works with signed integers and use it instead of the current one.

— RESOLUTION

Status: Pending.

MEDIUM **M-06**

PENDING

updatePeriod Reverts When No Votes Are Cast

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Minter.sol#L144
REVIEW	Main Review

— DESCRIPTION

The `updatePeriod` function in `Minter` handles the distribution of protocol emissions, including:

- Base emissions to `voter` for gauge rewards
- Growth emissions to ve holders
- Team emissions as a percentage of new emissions

However, if no votes have been cast, `totalWeight` in the `voter` contract will be zero. This causes a division-by-zero revert in `notifyRewardAmount` when attempting to distribute base emissions.

While uncommon, this can occur in the early stages of the protocol or with minimal voter participation.

When triggered, all emissions — including growth and team allocations — are blocked.

— RECOMMENDATION

Add a check to skip `voter` emissions when `totalWeight == 0`, allowing the remaining distributions to proceed uninterrupted.

— RESOLUTION

Status: Pending.

MEDIUM **M-07**

PENDING

Initial veDistributor rewards not claimable

CATEGORY	Rewards
LOCATION	VotingEscrowDistributor.sol
REVIEW	Main Review

DESCRIPTION

Each week the `veDistributor` receives emissions which it distributes to the `veEscrow` lockers. The reward claimable depends on the `bias` the locker holds at the very beginning of the given week. Because users won't have any locked balance at the very first week, nobody will be able to claim the rewards for the first 7 days.

RECOMMENDATION

To rescue the funds, you can add a check in `VotingEscrowDistributor` if the received rewards are for the very first epoch and refund them to the `treasury`.

```
``diff
else {
function checkpointToken() {
...
+ if (thisWeek == startTime) {
+ uint256 amount = (toDistribute * (nextWeek - t)) / sinceLast;
+ psp.transfer(treasury, amount);
+ psp.lastBalance -= amount;
+ continue;
+ }
tokensPerWeek[thisWeek] += (toDistribute * (nextWeek - t)) / sinceLast;
}
...
}
```

RESOLUTION

Status: Pending.

MEDIUM **M-08**

PENDING

Merge Of Expired Locks Breaks Claims

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L272
REVIEW	Main Review

— DESCRIPTION

The `merge` function allows combining two lock NFTs even if one is expired. This leads to unexpected behavior in reward distribution during claims.

Example scenario:

1. Alice creates Lock A for 3 weeks and lets it expire without claiming rewards.
2. Two weeks after Lock A expires, she creates Lock B and merges it into Lock A.
3. When claiming rewards with Lock A, `_claimable` iterates over the entire range between the original and new locks — including the 2-week gap where no lock existed.

Two unexpected outcomes were observed during testing:

1. The `balanceOf` calculation can underflow due to decay over periods with no active lock. This interacts with the unsafe casting issue reported separately. [VotingEscrowDistributor.sol#L318] (<https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/blob/main/contracts/VotingEscrowDistributor.sol#L318C17-L318C111>)
2. If no users were staked during gap epochs, `veSupply == 0`, the calculation of `toDistribute` reverts due a division by zero. [VotingEscrowDistributor.sol#L324] (<https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/blob/main/contracts/VotingEscrowDistributor.sol#L324>)

— RECOMMENDATION

Restrict `merge` to only allow combining of non-expired locks.

PROOF OF CONCEPT <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/pull/5/files>

— RESOLUTION

Status: Pending.

MEDIUM **M-09**

PENDING

Incorrect initialLastPoint Copy Breaks Snapshot

CATEGORY	Logical Error
LOCATION	https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L827
REVIEW	Main Review

DESCRIPTION

In the `_checkpoint` function, the local variable `initialLastPoint` is intended to store a snapshot of the state at the time of the last checkpoint. It is initialized as follows:

```
Point memory initialLastPoint = lastPoint;
```

However, due to Solidity's memory semantics for structs, this line does not create an independent copy. Instead, both `initialLastPoint` and `lastPoint` refer to the same memory location. As a result, any changes made to `lastPoint` are unintentionally reflected in `initialLastPoint`.

This shared reference leads to a logic error in the following line:

```
lastPoint.blk = initialLastPoint.blk + (blockSlope * (tI - initialLastPoint.ts)) /  
MULTIPLIER;
```

Here, `tI - initialLastPoint.ts` is supposed to represent the time elapsed since the last checkpoint. But since `initialLastPoint.ts` has already been mutated via changes to `lastPoint`, the difference is always zero. Consequently, `lastPoint.blk` does not advance as it should — it remains constant across all epochs. Every epoch ends up inheriting the same block number, regardless of when it was created.

Impact:

- `_findBlockEpoch` always returns the latest epoch, regardless of the target block.
- `totalSupplyAt` and `balanceOfAtNFT` return incorrect results for historical block queries.
- Snapshot voting relying on historical ve balances is broken.

RECOMMENDATION

Ensure a true copy of `lastPoint` is made when initializing `initialLastPoint`. This can be done manually by copying each field:

```
```plain text
```

```
Point memory initialLastPoint = Point({
 bias: lastPoint.bias,
 slope: lastPoint.slope,
```

**M-09** Incorrect initialLastPoint Copy Breaks Snapshot

CONTINUED

```
ts: lastPoint.ts,
blk: lastPoint.blk
});
...
```

**— RESOLUTION**

Status: Pending.

MEDIUM **M-10**

PENDING

# Loss Of Unclaimed Rewards And Bribes

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L285">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L285</a> , <a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L1051">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L1051</a>
REVIEW	Main Review

## DESCRIPTION

When veNFT holders call `merge` or `withdraw`, their tokens are burned without first claiming pending rewards from the Distributor contract. If they later attempt to claim, the `claim` function reverts because it tries to call `depositFor` on a non-existent (burned) token.

Even if `depositFor` were skipped, the contract would still not be able to determine the rightful recipient of the rewards, as the owner of the burned token cannot be retrieved.

Similarly, the bribe rewards for these users are lost as well, as the `getReward` function checks `ve.isApprovedOwner(msgs.sender, tokenId)` for the non-existent token.

## RECOMMENDATION

Add these calls:

- `Distributor.claim(tokenId)`
- `Bribe.getReward`

within both `merge` and `withdraw` functions, prior to burning, to guarantee that rewards and bribes are properly settled before burning the token.

## RESOLUTION

Status: Pending.

MEDIUM **M-11**

PENDING

# Griefing Via Bribe Rewards Token Length

CATEGORY DoS

LOCATION <https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Bribe.sol#L122>

REVIEW Main Review

## DESCRIPTION

The `notifyRewardAmount` function includes a check to ensure that the number of reward tokens does not exceed the `MAX_REWARD_TOKENS` limit, which is set to 16. This limit is intended to prevent unbounded growth of the `rewards` array.

However, in the current test configuration, `permissionMode` in `Voter.sol` is set to `false`, which allows anyone to whitelist a token as a reward token. As a result, a malicious actor can grief the system by depositing arbitrary or valueless tokens into `Bribe.sol`, filling up the rewards array and preventing legitimate reward tokens from being added.

## RECOMMENDATION

For production deployments, set `permissionMode = true` to ensure that only authorized accounts can whitelist reward tokens.

Consider also using (or removing) the `isBribe` mapping in `Voter` as it is currently able to be set by admin, but is not used anywhere.

## RESOLUTION

Status: Pending.

MEDIUM **M-12**

PENDING

# killGauge May Reset Unsent Rewards

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L426">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L426</a>
REVIEW	Main Review

## DESCRIPTION

The `killGauge` function can be called by an admin to disable a gauge which calls `distribute` to send out any remaining rewards before setting `isLive[gauge] = false` and resetting `claimable[gauge]` to zero.

The problem is that inside `distribute`, if the `claimable` amount is less than `IGauge(_gauge).left(psp)` or if `_claimable / DURATION` rounds to zero, the funds are not sent. Despite this, `killGauge` will still reset the `claimable` amount to zero. Consequently, any remaining funds not distributed will be stuck in the Voter contract.

## RECOMMENDATION

Modify the `killGauge` function to send any non-zero claimable amount to a specified address before resetting the variable.

## RESOLUTION

Status: Pending.

MEDIUM **M-13**

PENDING

# Lost emissions due to killed gauges

CATEGORY Rewards  
LOCATION Voter.sol  
REVIEW Main Review

## DESCRIPTION

When a gauge is killed, it stops accruing rewards in the `Voter`, however its weight is still contributing towards `totalWeight`. The received emissions in `Voter.notifyRewardAmount()` are divided by `totalWeight` to calculate the new reward index. Because killed gauges are also counted, the rest of the gauges receive less rewards than they should and the surplus remains locked in the contract, leading to a loss of emissions and value for the gauges.

## RECOMMENDATION

You can add a `killedWeight` variable which is subtracted from `totalWeight` in `notifyRewardAmount()`. The `killedWeight` variable will be increased when a gauge is killed and when someone votes for a killed gauge and it will be decreased when a gauge is revived and when someone resets a vote from a killed gauge. In result, the killed gauges won't be considered during reward distribution and the whole amounts will be claimable.

## RESOLUTION

Status: Pending.

MEDIUM **M-14**

PENDING

# Killed gauge may accrue rewards

CATEGORY Unexpected Behavior

LOCATION Voter.sol#L438

REVIEW Main Review

## — DESCRIPTION

`Voter.killGauge()` allows the admin to kill a gauge. Once killed, the gauge should stop accruing rewards. Whenever `_updateFor(gauge)` is executed, the gauge's global reward index is updated, but its `claimed` mapping is left unchanged if killed. By updating the index, but not the amount claimable, the gauge is successfully skipped.

However, `reviveGauge()` doesn't call `_updateFor(gauge)` before reviving the gauge. This creates a period of time between the last update for the killed gauge and the revival timestamp which is eligible for rewards. It's even possible that the gauge hasn't been updated for the whole duration from killing to reviving. In result, the killed gauge will receive from the rewards of the other gauges.

## — RECOMMENDATION

Update the gauge before revival

```
```diff
function reviveGauge(address _gauge) external onlyAdmin {
  require(!isLive[_gauge], ErrGaugeAlreadyLive());
  + _updateFor(_gauge);

  isLive[_gauge] = true;
  emit GaugeRevived(_gauge);
}
```
```

PROOF OF CONCEPT <https://gist.github.com/ZdravkoHr/6610a65a1595930e879aca2c0f6eed92>

## — RESOLUTION

Status: Pending.

MEDIUM **M-15**

PENDING

# Enabling autocompounding breaks the system

CATEGORY Configuration

LOCATION Global

REVIEW Main Review

## DESCRIPTION

The [PurrLP](https://purrsec.com/address/0xe5683f4bd410ea185692b5e6c9513be6bf1017ec/contract) contract has its `AUTO_COMPOUND_LP` set to `true` and there is no call to `setAutoCompoundFees` in the deployment script. This is a problem because the `Gauge` relies on non-compounding fees to be claimed and send as rewards to the `Bribe`. If `AUTO_COMPOUND_LP` stays `true`, the fees will be rewarded to the token holders, which means that LPs can deposit their tokens in the `Gauge` and receive both weekly emissions and swap fees, effectively stealing from the gauge voters.

## RECOMMENDATION

You can change `_AUTO_COMPOUND_FEES_` to `false`, but this solution removes the intended configuration flexibility of the pool.

Instead, a better approach is to change `PurrLP` to have autocompounding per pool and not globally. Then you can use the pools with no autocompounding in the ve3,3 system while still being able to have modifiable pools.

## RESOLUTION

Status: Pending.

MEDIUM **M-16**

PENDING

# Ineffective Burn of PSPS Tokens in burn

|          |                                                                                                                                                                                                                                             |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                               |
| LOCATION | <a href="https://github.com/GuardianOrg/purrswap-contractspurrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L248">https://github.com/GuardianOrg/purrswap-contractspurrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L248</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                 |

## — DESCRIPTION

The `burn` function in `VotingEscrow` is designed to permanently destroy a veNFT and the underlying locked PSPS tokens. However, it does so by transferring the locked PSPS tokens to the zero address:

```
require(IERC20(token).transfer(address(0), value), ErrTransferFailed());
```

Transferring tokens to the zero address does not reduce the reported `totalSupply` of PSPS, creating a discrepancy between the actual circulating supply and the on-chain total supply.

As a result:

- PSPS emissions are over-calculated by the Minter, since they are based on an inflated `totalSupply`
- Market cap and FDV calculations become distorted, as many analytics platforms rely on `totalSupply` to compute these metrics, potentially impacting token price.

Additionally, the `burn` function appears non-essential and introduces a potential footgun if mistakenly called.

## — RECOMMENDATION

If token burning is intended, modify the function to properly reduce PSPS total supply by exposing and calling a proper `_burn` function in the PSPS token contract.

Else, consider removing the `burn` function from `VotingEscrow` entirely to reduce complexity and eliminate potential misuse.

## — RESOLUTION

Status: Pending.

MEDIUM **M-17**

PENDING

# Precision Loss Leads To Irrecoverable Rewards

CATEGORY Rounding

LOCATION <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/blob/main/contracts/Gauge.sol#L285>

REVIEW Main Review

## — DESCRIPTION

The `notifyRewardAmount` function in `Gauge.sol` calculates the reward rate for a token as:

```
rewardRate[token] = amount / DURATION;
```

This calculation truncates any remainder resulting in dust amounts of tokens being trapped in the contract. Assuming WBTC is the reward token, for 1 WBTC deposited, 0.000026 WBTC (approximately \$3) is stuck. This unclaimable amount grows each week with every new reward deposit, across all reward tokens, resulting in a material loss over time.

## — RECOMMENDATION

Implement a `recover` function, callable by an admin, to allow manual recovery of accumulated dust.

PROOF OF CONCEPT <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-team1/pull/3/files>

## — RESOLUTION

Status: Pending.

MEDIUM **M-18**

PENDING

# Restricted poke call frequency

CATEGORY Unexpected Behavior

LOCATION Voter.sol

REVIEW Main Review

## DESCRIPTION

The `Voter.poke()` function can be called once for each voting period, which is 7 days long. This is different from some other protocols that allow calling `poke()` at any point in time. Because of this, the Voter will be using outdated weight of the user. For example, if emissions happen on day 0-7-14...., and two users create the exact same locks on day 1, but user A votes immediately while user B votes on the 6th day, user B will have less voting power than user A and user A cannot be poked because their token is locked until day 8. This will result in them directing bigger part of the emissions towards their desired gauge on day 7 when the actual distribution happens.

## RECOMMENDATION

Consider changing `poke()` in such a way that it doesn't require the token to be expired by calling `isVoteExpired`.

## RESOLUTION

Status: Pending.

MEDIUM **M-19**

PENDING

# Incorrect Epoch Accounting for Rewards

|          |                                                                                                                                                                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                               |
| LOCATION | <a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/bb94a935ec97ff65b8c91a38cb98a864f637e535/contracts/Bribe.sol#L197">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team2/blob/bb94a935ec97ff65b8c91a38cb98a864f637e535/contracts/Bribe.sol#L197</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                 |

## — DESCRIPTION

The `earned` function in the Bribe contract calculates the amount of rewards earned per token per epoch. The issue lies in how the function determines the token balance and total supply for a given epoch: it uses `_currTs + DURATION` as the timestamp, which corresponds to the very first second of the next epoch. As a result, any deposits made at the beginning of the next epoch are incorrectly included in the previous epoch. This is problematic because users who did not have a previous deposit will not be able to claim that epoch's rewards, yet their deposited tokens are still factored into the reward calculation. Consequently, those rewards become unclaimable and remain stuck in the contract.

## — RECOMMENDATION

Adjust the timestamp used in the `earned` function to reflect the actual end of the epoch (e.g., `_currTs + DURATION - 1`) to ensure only balances from the correct epoch are used in reward calculations.

## — RESOLUTION

Status: Pending.

MEDIUM **M-20**

PENDING

# Wrong reward validation in Bribe

CATEGORY Validation  
LOCATION Bribe.sol#L121  
REVIEW Main Review

## DESCRIPTION

`Bribe.notifyRewardAmount()` validates the sent reward token is whitelisted by calling `Voter.isWhitelisted()`. This is not the correct validation because whitelisted tokens are tokens that are allowed to be used as base and quote for creating new pools. There is a separate mapping in the `Voter` contract called `isBribe` that's used to enable bribe rewards.

The current implementation is problematic because it leads to unexpected behaviors. For instance:

1. A few pools are created with tokens A and B. The voter admin wants to limit the creation of these pools, so he delists the two tokens, but he still wants to support the already created ones. Because `Bribe` checks if the tokens are whitelisted, it won't be able to receive rewards anymore.
2. The voter admin wants to stop a bribe from receiving a particular token as reward so he sets `isBribe` to `false`, but this doesn't affect the Bribe and it can continue accruing such rewards.

## RECOMMENDATION

Check for `isBribe` instead

```
`diff
- require(voter.isWhitelisted(token), ErrTokenNotWhitelisted());
+ require(voter.isBribe(address(this), token), ErrTokenNotWhitelisted());
`
```

## RESOLUTION

Status: Pending.

LOW **L-01**

PENDING

## Duplicate read from storage

LOCATION `https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/VotingEscrow.sol#L207`

REVIEW Main Review

### — DESCRIPTION

The `approve()` function performs two reads from the same storage slot.

### — RECOMMENDATION

Use the `owner` variable instead of reading from storage a second time.

### — RESOLUTION

Status: Pending.

LOW **L-02**

PENDING

## Malicious Admin can frontrun whitelist()

LOCATION `https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L189-L197`

REVIEW Main Review

### — DESCRIPTION

When a user calls `whitelist()`, they are required to pay a `listingFeeParameters.fee` in the PSPS token. Since this value is not specified in the function call and the admin can update the fee value at any time, a malicious admin can frontrun the call and increase the amount. A user with a larger allowance of PSPS tokens could suffer losses.

### — RECOMMENDATION

Create an `expectedFee` parameter and check it against the value of `listingFeeParameters.fee` in storage. Revert the call if they differ.

### — RESOLUTION

Status: Pending.

LOW **L-03**

PENDING

## Tokens that are delisted will remain active in existing gauges

LOCATION <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L199-L203>

REVIEW Main Review

### — DESCRIPTION

When the admin delists a token, any gauges that are already configured with this token will remain active.

### — RECOMMENDATION

Consider adding a function to kill all gauges the rely on a specific token when delisting the token.

### — RESOLUTION

Status: Pending.

LOW **L-04**

PENDING

# Permissionless pool and gauge deployments can DoS updates and distributions

LOCATION <https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L261-L263>

REVIEW Main Review

## — DESCRIPTION

Given that the deployments of pools and gauges are permissionless, a single actor can deploy numerous pools and gauges, causing the function to update and distribution for all gauges to fail due to an Out-Of-Gas error.

## — RECOMMENDATION

This is already mitigated due to the presence of the individual `updateFor()` and `updateForRange()` functions. However, `updateForRange()` is somewhat compromised by the over-deployment of gauges as well. The project team will need to maintain a list of legitimate pools and update through the `updateFor(address[] memory _gauges)` function.

The distribution functions will require the same monitoring as well to effectively distribute to all gauges.

## — RESOLUTION

Status: Pending.

LOW **L-05**

PENDING

# Inconsistent Flash-Loan Block Protection Across balanceOfNFT Variants

|          |                       |
|----------|-----------------------|
| CATEGORY | Logical Error         |
| LOCATION | VotingEscrow.sol: 504 |
| REVIEW   | Main Review           |

## — DESCRIPTION

In the `VotingEscrow` contract, the `balanceOfNFT(uint256 _tokenId)` function enforces a flash-loan protection preventing newly transferred veNFTs from being used for voting or gauge boosting in the same block by returning zero if `ownershipChange[_tokenId] == block.number`. However, the other `balanceOfNFT(uint256 _tokenId, uint256 _t)` and `balanceOfAtNFT(uint256 _tokenId, uint256 _block)` functions that compute the NFT's voting power do not enforce the same ownership-change check. As a result, protocols or external calls that rely on them can see a non-zero voting balance even if the veNFT was transferred in the current block.

## — RECOMMENDATION

Align the same-block flash check in the `balanceOfNFT(uint256 _tokenId, uint256 _t)` and `function balanceOfAtNFT(uint256 _tokenId, uint256 _block)` functions.

## — RESOLUTION

Status: Pending.

LOW **L-06**

PENDING

# Incorrect view function `listingFee()`

LOCATION `https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L369C5-L371C6`

REVIEW Main Review

## DESCRIPTION

`listingFee()` is supposed to return amount of PSPS required to list a new token by performing the following calculation:

```
``solidity
(IERC20Metadata(psp).totalSupply() - IERC20Metadata(ve).totalSupply()) /
listingFeeParameters.fee;
`
```

However, we see in the `whitelist()` function that the true amount taken from the user in order to list a new token is simply `ListingFeeParameters.fee`:

```
``solidity
function whitelist(address _token) public checkPermissionMode {
 if (msg.sender != admin) {
 if (listingFeeParameters.fee > 0) {
 psp.safeTransferFrom(msg.sender, listingFeeParameters.recipient, listingFeeParameters.fee);
 }
 }

 _whitelist(_token);
}
`
```

## RECOMMENDATION

Either use the public `listingFee()` view function in the `safeTransferFrom()` call or alter the view function to simply return the `ListingFeeParameters.fee` value.

## RESOLUTION

Status: Pending.

LOW **L-07**

PENDING

## Two files with the same name

CATEGORY Best Practices  
LOCATION contracts/interfaces  
REVIEW Main Review

### — DESCRIPTION

There are currently two files with the same name, but different case -

`contracts/interfaces/IPurrLp0racle.sol` and `contracts/interfaces/IPurrLP0racle.sol`. The only difference is the `LP` and `Lp`. The file used by the system is `LP`, so the other one can be deleted. The existence of such a file may cause problems on case-insensitive operating systems like MacOS and Windows - for example `IPurrLp0racle.sol` overrides `IPurrLP0racle.sol` when the repository is cloned.

### — RECOMMENDATION

Delete the `IPurrLp0racle.sol` file

### — RESOLUTION

Status: Pending.

LOW **L-08**

PENDING

# VotingEscrow is not ERC721 compliant

CATEGORY Unexpected Behavior  
LOCATION VotingEscrow.sol  
REVIEW Main Review

## DESCRIPTION

The `VotingEscrow.supportsInterface()` function returns `true` when it's called with the interface id for ERC721 (0x80ac58cd). However, it doesn't implement it correctly because:

- `balanceOf(address)` doesn't revert when called with `address(0)`.
- `ownerOf()` doesn't revert for invalid ids.
- `transferFrom()` and `safeTransferFrom()` don't revert when NFTs are transferred to the 0 address.
- `getApproved(uint256)` doesn't revert when it receives invalid token id.

## RECOMMENDATION

Fix the 4 issues from the report to make the contract fully compliant.

## RESOLUTION

Status: Pending.

LOW **L-09**

PENDING

## Duplicate function `distro()` can be removed

**LOCATION** <https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L333-L335> (<https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L333-L339>)

**REVIEW** Main Review

### — DESCRIPTION

There are two functions `distro()` and `distribute()` that are identical. However, `distribute()` is used in the `Gauge.sol` contract.

### — RECOMMENDATION

Remove the `distro()` function.

### — RESOLUTION

Status: Pending.

LOW **L-10**

PENDING

## User can create false emitWithdraw emissions

LOCATION `https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Gauge.sol#L260`

REVIEW Main Review

### — DESCRIPTION

A user can deposit stake token into `Gauge.sol` and attach a VE token id. This will call `IVoter(voter).emitDeposit(tokenId, msg.sender, amount);` signaling that the user has deposited the specific amount with the specified token id.

However, a user is able to withdraw their token id without withdrawing their stake token via `withdrawToken()`. They are not required to withdraw their stake token.

The user can then deposit another token id and withdraw all of their stake token while withdrawing the new token id. This will call `IVoter(voter).emitWithdraw(tokenId, msg.sender, amount);` signaling that the new token id withdrew the large stake amount, however, the stake was deposited with the previous token. This may present an inaccurate history of deposits and withdrawals based on token ids.

### — RECOMMENDATION

When withdrawing the token, require that the user withdraw all of their stake.

### — RESOLUTION

Status: Pending.

LOW **L-11**

PENDING

# Minter initializing can be blocked

CATEGORY Informational

LOCATION Minter.sol

REVIEW Main Review

## — DESCRIPTION

When `Minter.initialize()` calls `createLockFor()`, the execution flow will be given to each NFT recipient. If the recipient can't receive the NFT, either because the account doesn't support NFTs or on purpose, the initialize will fail and it has to be executed again.

## — RECOMMENDATION

Keep that possibility in mind.

## — RESOLUTION

Status: Pending.

LOW **L-12**

PENDING

## Minter doesn't follow the CEI pattern

CATEGORY Best Practices

LOCATION Minter.sol

REVIEW Main Review

### — DESCRIPTION

The `Minter.initialize()` function loops through all the lockers, calls `ve.createLockFor()` for each one of them and sets the state of the contract only after that. Since `createLockFor()` will mint NFT, the execution flow will be given to the NFT recipient. There is no direct risk for the contract, but it's possible that a read-only reentrancy happens if any of the other contracts read from `Minter` in the future.

### — RECOMMENDATION

Consider moving the for loop to the end of the `initialize()` function.

### — RESOLUTION

Status: Pending.

LOW **L-13**

PENDING

# Inaccurate Minter MAX\_LOCK

CATEGORY Math  
LOCATION Minter.sol#L26  
REVIEW Main Review

## DESCRIPTION

The Minter contract has defined a `MAX_LOCK` constant which should be equal to 4 years, the maximum time tokens can be locked for in the voting escrow. However, the value of this constant is `52 * 4 weeks` which is `125798400`. In contrast, 4 years in the escrow are defined as `4 * 365 * 86400 = 126144000`. The locks created by the minter are with 4 days shorter. Because the timestamp is rounded to the nearest week, this results in a week difference than the actual max lock.

## RECOMMENDATION

```
``diff
- uint256 public constant MAX_LOCK = 52 * 4 weeks; // 4 years
+ uint256 public constant MAX_LOCK = 4 * 365 * 86400
``
```

## RESOLUTION

Status: Pending.

LOW **L-14**

PENDING

# listingFee Reverts If Fee Not Set

|          |                                                                                                                                                                                                                                 |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Unexpected Behavior                                                                                                                                                                                                             |
| LOCATION | <a href="https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Voter.sol#L370">https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Voter.sol#L370</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                     |

## DESCRIPTION

The public view function `listingFee` computes a fee by dividing by `listingFeeParameters.fee`. Since `listingFeeParameters.fee` defaults to zero, any call to `listingFee` before an admin invokes `setListingFeeParameters` will revert with a division-by-zero error. Front-end integrations expecting a numeric fee (even zero) will instead see unexpected reverts or broken UI.

## RECOMMENDATION

Guard against a zero denominator by returning 0 when `listingFeeParameters.fee` is unset or zero.

## RESOLUTION

Status: Pending.

LOW **L-15**

PENDING

# initialize Overrides Intended Emissions Delay

|          |                                                                                                                                                                                                                                       |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                         |
| LOCATION | <a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Minter.sol#L198">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Minter.sol#L198</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                           |

## — DESCRIPTION

In the constructor, `activePeriod` is set to defer emissions by 2 weeks, aligning with the comment:

```
// The activePeriod is set to start 2 weeks from deployment (rounded to week boundary)
```

However, this delay is overwritten in the `initialize` function with:

```
activePeriod = (block.timestamp / WEEK) * WEEK;
```

This resets `activePeriod` to the current week, effectively removing the intended 2-week delay. As a result, emissions can begin after just one week instead of two.

## — RECOMMENDATION

Ensure `initialize` preserves the constructor-set `activePeriod` or re-implements the 2-week offset if `initialize` is intended to control it.

## — RESOLUTION

Status: Pending.

LOW **L-16**

PENDING

# Rewards claim underflows for invalid tokens

CATEGORY Unexpected Behavior  
LOCATION VotingEscrowDistributor.sol#L274  
REVIEW Main Review

## DESCRIPTION

`VotingEscrowDistributor._claim()` takes the `userEpoch` returned by `_claimable()` and subtracts 1 from it. However, `_claimable()` returns 0 if `maxUserEpoch == 0`. Because it's an `uint`, the transaction will revert with an `underflow`.

This may be unexpected, especially for a call to `claimMany()`.

## RECOMMENDATION

You can implement the following change to avoid reverts:

```
`diff
if (maxUserEpoch == 0) {
- return (0, 0, 0, 0);
+ return (0, 0, type(uint256).max, 0);
}
```

If reverts are the desired outcome, you can then check if `userEpoch == type(uint256).max` in the `_claim()` function and revert with a descriptive error message.

## RESOLUTION

Status: Pending.

LOW **L-17**

PENDING

# Rewards can be recovered from MultiRewards

CATEGORY Rewards  
LOCATION MultiRewards.sol  
REVIEW Main Review

## DESCRIPTION

`MultiRewards.recover()` allows the owner to rescue ERC20 tokens from the contracts, including reward tokens. This feature is useful for recovering reward tokens stuck in the contract due to the precision loss in `notifyRewardAmount` when calculating the `rate`, but since there is no limit set to the amount to be recovered, the contract may be put in a state where it doesn't have enough funds to pay out rewards.

## RECOMMENDATION

Consider adding a `dust` field to the `Reward` struct and update it in `notifyRewardAmount()`

```
``diff
_rewardData[rewardToken].rewardRate = amount / _rewardData[rewardToken].rewardsDuration;
_rewardData[rewardToken].lastUpdateTime = block.timestamp;
_rewardData[rewardToken].periodFinish = block.timestamp + _rewardData[rewardToken].rewardsDuration;
+ uint256 claimableAmount = _rewardData[rewardToken].rewardRate *
_rewardData[rewardToken].rewardsDuration;
+ _rewardData[rewardToken].dust += amount - claimableAmount
``
```

Then don't allow pulling more reward tokens than `dust`

```
``diff
function recover(address _token, address _to, uint256 _amount) external onlyOwner {
if (_token == stakingToken && _amount > stakingToken.balanceOf(address(this)) - totalSupply) {
revert ErrSkimmingTooMuch();
}

+ if (_rewardData[_token].lastUpdateTime != 0) {
+ if (amount > _rewardData[_token].dust) {
+ revert ErrSkimmingTooMuch();
```

**L-17** Rewards can be recovered from MultiRewards

CONTINUED

```
+ }
+ _rewardData[_token].dust -= amount;
+ }

if (_token == address(0)) {
 SafeTransferLib.safeTransferETH(_to, _amount);
} else {
 _token.safeTransfer(_to, _amount);
}

emit LogRecovered(_token, _to, _amount);
}
...
```

You can also implement the `residualAmount` approach where any dust is added towards the next rewards distribution.

**— RESOLUTION**

Status: Pending.

LOW **L-18**

PENDING

## Points can leave not whitelisted wallets

|          |             |
|----------|-------------|
| CATEGORY | Validation  |
| LOCATION | Points.sol  |
| REVIEW   | Main Review |

### — DESCRIPTION

`Points` is an `ERC20` token which can be transferred only from or to a whitelisted address. This restriction is not applied if `from == address(0)` or `to == address(0)` to allow minting and burning. However, the token cannot be burned, since there is no exposed function in the ABI which calls `_burn()`. On the other hand, users can call `transfer` with `address(0)` as recipient. The transaction will then go through and the tokens will leave their wallets.

The users can also send them to `MultiRewards` without staking, but in this case they can be pulled out of it by calling `recover()`.

### — RECOMMENDATION

If you want to prevent tokens leaving their wallets entirely and all the whitelisted addresses will be initiating the transfers, you can change the `_beforeTransferFunction` to

```
``plain text
function _beforeTokenTransfer(address from, address to, uint256 /*amount*/) internal view
override {
 if (from != address(0) && !whitelisted[msg.sender]) {
 revert NotAllowed();
 }
}
```

### — RESOLUTION

Status: Pending.

LOW **L-19**

PENDING

## Part of MultiRewards may not be claimed

|          |                  |
|----------|------------------|
| CATEGORY | Rewards          |
| LOCATION | MultiRewards.sol |
| REVIEW   | Main Review      |

### — DESCRIPTION

If `MultiRewards.notifyRewardAmount()` is called when there are no active stakers, any amount from that moment until the first stake happens will not be claimable by anyone. The current implementation allows rescuing reward tokens, but if [this finding](https://www.notion.so/guardianaudits/L-07-1ff8bda5828c818a974cc1a0f316767f?pvs=4) is fixed, the rewards will remain stuck there.

### — RECOMMENDATION

Plan carefully when to call `notifyRewardAmount()`

### — RESOLUTION

Status: Pending.

LOW L-20

PENDING

# No rewards length cap in MultiRewards

CATEGORY Rewards  
LOCATION MultiRewards.sol  
REVIEW Main Review

## DESCRIPTION

`MultiRewards` doesn't implement any limit for reward tokens length. This makes it possible to push a lot of reward tokens in the `rewards` array by calling `addReward()` and potentially causing the for loop in `_updateRewards()` to revert with OOG. Since `_updateRewards()` is called on each interaction, this will result in a DOS of the contract.

## RECOMMENDATION

Consider implementing a length limit for reward tokens.

## RESOLUTION

Status: Pending.

LOW **L-21**

PENDING

# MultiRewards tokens can be blacklisted

|          |                  |
|----------|------------------|
| CATEGORY | Rewards          |
| LOCATION | MultiRewards.sol |
| REVIEW   | Main Review      |

## DESCRIPTION

The `MultiRewards.setRewardsDuration()` function allows changing the `rewardsDuration` of a reward token to a non-zero amount, even if the token has not been added.

This behavior can be used to permanently disable adding a given token as a reward, since `addReward()` requires that the current `rewardsDuration` for the new token is 0.

If the blacklisting is used, 2 new problems emerge:

- `notifyRewardAmount()` can be called for the modified reward token. The `rewardRate`, `lastUpdateTime` and `periodFinish` fields will be updated, but since the token is not part of `rewardToken`, the amount will never be paid out.
- `isSupportedReward()` will return `true` for the blacklisted token since it relies on `rewardsDuration`

## RECOMMENDATION

If such blacklisting behavior is desirable, you need to adjust `isSupportedReward()` and `notifyRewardAmount`. If you implement the `scalingFactor` fix from [this issue] (/1ff8bda5828c810b8078ccf1d7287939?pvs=25), the functions can be changed to use it instead of `rewardsDuration`.

```
``diff
function isSupportedReward(address rewardToken) external view returns (bool) {
- return _rewardData[rewardToken].rewardsDuration != 0;
+ return _rewardData[rewardToken].scalingFactor != 0;
}
`
``diff
function notifyRewardAmount(address rewardToken, uint256 amount) external
onlyOwnerOrRoles(ROLE_REWARD_DISTRIBUTOR) {
- if (_rewardData[rewardToken].rewardsDuration == 0) {
+ if (_rewardData[rewardToken].scalingFactor == 0) {
revert ErrInvalidTokenAddress();
}
```

**L-21** MultiRewards tokens can be blacklisted

CONTINUED

```

...
}
``

The alternative and more concise solution is to simply not allow changing the rewardsDuration
for tokens that are not added.
`diff
function setRewardsDuration(address rewardToken, uint256 _rewardsDuration) external onlyOwner {
if (block.timestamp <= _rewardData[rewardToken].periodFinish) {
revert ErrRewardPeriodStillActive();
}
if (_rewardsDuration == 0) {
revert ErrZeroDuration();
}
+ if (_rewardData[rewardToken].rewardsDuration == 0) {
+ revert ErrInvalidTokenAddress();
+ }
_rewardData[rewardToken].rewardsDuration = _rewardsDuration;
emit LogRewardsDurationUpdated(rewardToken, _rewardData[rewardToken].rewardsDuration);
}
``

```

**— RESOLUTION**

Status: Pending.

LOW **L-22**

PENDING

# No Removal Function for Inactive Reward Tokens

CATEGORY Best Practices

LOCATION <https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/MultiRewards.sol#L128>

REVIEW Main Review

## — DESCRIPTION

The MultiRewards contract loops through the entire `rewardTokens` array each time a user claims rewards. While the owner can add new reward tokens via `addReward` and effectively "disable" them by setting their duration to zero using `setRewardsDuration`, these deactivated tokens are not removed from the array. As a result, the array of reward tokens can grow indefinitely. Over time, this leads to unnecessary iterations during reward claims, increasing gas costs for users. In extreme cases, this could cause out-of-gas errors and make the contract unusable for claims.

## — RECOMMENDATION

Implement a `removeToken` function to allow the owner to clean up inactive reward tokens from the array.

## — RESOLUTION

Status: Pending.

LOW **L-23**

PENDING

# Operator Receives Funds Instead of veNFT Owner

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Unexpected Behavior                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L1048">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L1048</a> , <a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Bribe.sol#L109">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/Bribe.sol#L109</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## — DESCRIPTION

In both VotingEscrow and Bribe contracts, some functions are gated by the common pattern:

```
require(isApprovedOrOwner(msg.sender, tokenId), "Not authorized");
```

This allows an approved operator to act on behalf of a veNFT owner. However, functions like `withdraw` or `getReward` transfer funds to `msg.sender`, not the actual veNFT owner.

This behavior may be unexpected or incorrect in certain cases—especially if the operator is a contract that cannot safely receive tokens or if the operator was merely automating on behalf of the user.

## — RECOMMENDATION

Review whether this pattern is intentional. If not, consider changing the destination of transfers to `ownerOf(tokenId)` rather than `msg.sender`. This ensures that funds always return to the veNFT owner, regardless of who initiates the action.

## — RESOLUTION

Status: Pending.

LOW **L-24**

PENDING

## ownershipChange Can Be Bypassed With merge

|          |                                                                                                                                                                                                                                               |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                 |
| LOCATION | <a href="https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L272">https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/VotingEscrow.sol#L272</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                   |

### — DESCRIPTION

The `ownershipChange` mapping is intended to prevent flash attacks by disallowing voting in the same block a veNFT is minted or transferred.

However, this protection can be bypassed by minting a new veNFT and immediately merging it into an existing one. Since the `merge` function does not update `ownershipChange` for the destination `tokenId`, the merged veNFT can vote in the same block, defeating the purpose of the flash protection.

### — RECOMMENDATION

In the `merge` function, consider updating the `ownershipChange` mapping for the `to` `tokenId`.

### — RESOLUTION

Status: Pending.

LOW **L-25**

PENDING

# Ambiguous event emission

CATEGORY Events  
LOCATION Gauge.sol#L161  
REVIEW Main Review

## — DESCRIPTION

`Gauge.getReward()` can be called by users to claim their rewards, but also from the `Voter` contract to claim any user's rewards. However, the `ClaimRewards` event emits `msg.sender` as `from` instead of `account`. This will result in the `Voter` being emitted as `from`, which is ambiguous because the rewards are sent to `account`.

## — RECOMMENDATION

Consider whether you want the `from` field to be the account receiving the rewards or the account initiating the claim. You can also add a new field to the event to have both of them.

## — RESOLUTION

Status: Pending.

LOW **L-26**

PENDING

# Precision Loss Leaves Reward Dust

|          |                                                                                                                                                                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Rounding                                                                                                                                                                                                                                          |
| LOCATION | <a href="https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/MultiRewards.sol#L190">https://github.com/GuardianOrg/purrrswap-contracts/purrrswap-ve33-team1/blob/main/contracts/MultiRewards.sol#L190</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                       |

## — DESCRIPTION

In MultiRewards.sol, the `notifyRewardAmount` function sets the reward rate as follows:

```
_rewardData[rewardToken].rewardRate = amount / _rewardData[rewardToken].rewardsDuration;
```

This calculation uses integer division and truncates any remainder. As a result, a small amount of reward tokens (i.e., dust) remains unallocated and is not distributed to users.

## — RECOMMENDATION

While this does not result in permanent loss as the contract includes a `recover` function for ERC20 tokens, consider documenting this behavior for users and integrators to ensure transparency.

## — RESOLUTION

Status: Pending.

LOW **L-27**

PENDING

# Inconsistent veNFT Access Control in deposit

CATEGORY Access Control

LOCATION <https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Gauge.sol#L197>

REVIEW Main Review

## — DESCRIPTION

Across the system, interactions involving a veNFT typically check:

```
IVotingEscrow(ve).isApprovedOrOwner(msg.sender, _tokenId)
```

This allows approved operators to act on behalf of the veNFT owner, supporting delegated workflows.

However, in `Gauge.sol`, the `deposit` function enforces a stricter check:

```
require(IVotingEscrow(ve).ownerOf(tokenId) == msg.sender, ErrInvalidTokenId());
```

This hard requirement for the caller to be the exact owner blocks approved operators from depositing on the owner's behalf, breaking consistency with the rest of the system and limiting the utility of delegation.

## — RECOMMENDATION

Update `Gauge.sol` to use `isApprovedOrOwner` in the `deposit` function, aligning it with the access pattern used elsewhere in the system.

## — RESOLUTION

Status: Pending.

LOW **L-28**

PENDING

## Duplicate import statement

LOCATION `https://github.com/GuardianOrg/purrrswap-contractspurrrswap-ve33-fuzz/blob/d00d434216bc3484ab6f5e7d6c7c038659cf3f0a/contracts/Voter.sol#L11`

REVIEW Main Review

### DESCRIPTION

`IPurrFactory.sol` is imported twice.

### RECOMMENDATION

Remove the duplicate import.

### RESOLUTION

Status: Pending.

LOW **L-29**

PENDING

# First Epoch Emissions Over-Boosted

|          |                                                                                                                                                                                                                                   |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Informational                                                                                                                                                                                                                     |
| LOCATION | <a href="https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Minter.sol#L119">https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Minter.sol#L119</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                       |

## — DESCRIPTION

In Minter.sol, the `updatePeriod` function computes weekly emissions as follows:

```
```plain text
```

```
if (epochCount <= DECAY_EMISSION_DURATION_IN_EPOCHS) {  
    //@audit should "_emission = _weekly;" be moved here  
    if (epochCount <= BOOSTED_EMISSION_DURATION_IN_EPOCHS) {  
        _weekly = (_weekly * BOOSTED_EMISSION_IN_BIPS) / BIPS;  
    } else {  
        _weekly = (_weekly * DECAY_EMISSION_IN_BIPS) / BIPS;  
    }  
  
    _emission = _weekly;  
    weekly = _weekly;  
}  
`
```

This sets `_emission` after applying the boost or decay adjustment to `_weekly`. In contrast, other Solidity forks typically assign `_emission = _weekly` before any adjustment. This subtle difference results in boosted emissions during the first epoch, which can increase `_growth` and cause emissions to compound more than expected across epochs.

The result is a higher-than-intended inflation path, especially early in the distribution schedule, which may affect total supply projections and long-term tokenomics.

— RECOMMENDATION

Review whether the current ordering aligns with intended tokenomics. If the goal is for the first epoch to distribute unboosted emissions, consider assigning `_emission = _weekly` before applying any boost or decay.

— RESOLUTION

Status: Pending.

LOW **L-30**

PENDING

balanceOfAtNFT () is not fully accurate

CATEGORY	Warning
LOCATION	VotingEscrow.sol
REVIEW	Main Review

— DESCRIPTION

`VotingEscrow.balanceOfAtNFT ( )` increases the `blockTime` variable with the timestamp delta between the desired block and the last recorded block. Since it works with block numbers, it calculates the average block time difference (`dt / dBlock`) and multiplies it by the block delta.

While this approach should be close to the real voting power of the user at that block, it is not fully accurate because the time difference between the blocks may be more than 12 seconds, if a block was not proposed in a given epoch. For example, [block 22523350](https://etherscan.io/block/22523350) and [block 22523349](https://etherscan.io/block/22523349) have a gap of 24 seconds.

This is a bigger problem when `balanceOfAtNFT ( )` enters the `_epoch >= maxEpoch` case because both `dt` and `dBlock` are calculated using the current block. This means there may be difference when `balanceOfAtNFT ( )` is called onchain for the same block number, depending on when it is called.

— RECOMMENDATION

Document that:

1. `balanceOfAtNFT(tokenID, block)` is a close, but not exact value.
2. `balanceOfAtNFT(tokenID, block)` is non-deterministic, i.e it may return different result even if it's called with the same parameters.

— RESOLUTION

Status: Pending.

LOW **L-31**

PENDING

Wrong owner for Minter deployment

CATEGORY Configuration

LOCATION `ve.ts#L154`

REVIEW Main Review

DESCRIPTION

The `ve.ts` script first deploys the VeSystem contract then checks if the deployer's address is the same as the `OWNER` constant and if not, ownership is transferred to the `OWNER`. However, the `Minter` contract was deployed with `OWNER` as the owner. If the script is executed from any other deploying address besides `OWNER`, the deployment will fail because `Minter.initialize()` which has `onlyOwner` modifier is called by the deployer, while the actual owner is `OWNER`.

RECOMMENDATION

Pass the deployer as owner when deploying the minter

```
``diff
// ### 11. Deploy Minter
const minter = await deploy("Minter", {
  from: deployer.address,
  args: [
    voter.address,
    votingEscrow.address,
    veDistributor.address,
    - OWNER, // Owner address
    + deployer.address // Owner address
  ],
  log: true,
  waitConfirmations: 1,
  skipIfAlreadyDeployed: true,
  contract: "Minter",
});
``
```

L-31 Wrong owner for Minter deploymentCONTINUED

— RESOLUTION

Status: Pending.

LOW **L-32**

PENDING

Lack of artProxy

CATEGORY Configuration

LOCATION ve.ts

REVIEW Main Review

DESCRIPTION

`VotingEscrow.tokenURI()` calls the `artProxy` set by the owner to return the URI, but when `VotingEscrow` is deployed there is no `artProxy` set. In result all calls to `VotingEscrow.tokenURI()` will fail until such a proxy is set.

RECOMMENDATION

Consider setting an `artProxy` as soon as possible after deployment.

RESOLUTION

Status: Pending.

LOW **L-33**

PENDING

Only 2 tokens whitelisted

CATEGORY Informational

LOCATION ve.ts

REVIEW Main Review

— DESCRIPTION

Currently `ve.ts` whitelists only `WHYPE` and `PSPS`, meaning only pools for these two tokens can be created.

— RECOMMENDATION

Consider if there should be more tokens whitelisted.

— RESOLUTION

Status: Pending.

LOW **L-34**

PENDING

Incorrect Mapping and Detach Logic In Withdraw

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Gauge.sol#L244
REVIEW	Main Review

DESCRIPTION

In `Gauge.sol`, the `withdrawToken` function contains flawed logic that mishandles `tokenIds[msg.sender]` and improperly manages gauge attachment. When a user performs a partial withdrawal, the function unconditionally sets `tokenIds[msg.sender] = 0`, even if the user still has a staked balance. This causes a revert on subsequent withdrawals due to the `require(tokenId == tokenIds[msg.sender])` check. The only workaround is to call deposit again to restore the mapping, introducing friction and potential user confusion.

Additionally, two more issues were identified:

1. `IVoter(voter).detachTokenFromGauge` is called during partial withdrawals. This is incorrect, as detachment should only occur when the user fully exits the gauge.
2. If `withdrawToken` is called with `tokenId = 0`, it enters the `else` branch. In this path, the `tokenId` mapping is not cleared and `detachTokenFromGauge` is not called, leaving the veNFT in a stale attached state. This will later cause withdrawals in VotingEscrow to revert, since the token is still marked as attached to the gauge.

RECOMMENDATION

- Only reset `tokenIds[msg.sender]` and call `detachTokenFromGauge` if the user's balance after withdrawal is zero.
- Ensure `detachTokenFromGauge` is never called on partial withdrawals.
- In `withdrawToken`, validate the `tokenId` and handle the zero case properly to guarantee correct cleanup and detachment.

RESOLUTION

Status: Pending.

LOW **L-35**

PENDING

Duplicate Reward Tokens Possible via Frontrun

CATEGORY	Frontrunning
LOCATION	https://github.com/GuardianOrg/purrswap-contracts/purrswap-ve33-team1/blob/main/contracts/Gauge.sol#L508
REVIEW	Main Review

— DESCRIPTION

The Gauge contract lacks a dedicated mechanism to explicitly deactivate a reward token. Instead, admins must use the `swapOutReward` function, which sets `isReward[oldToken] = false` and `isReward[newToken] = true`. However, this design introduces a race condition that can be exploited by a griefer. Specifically, an attacker can frontrun the admin's `swapOutReward` transaction and call `notifyRewardAmount` with a minimal amount (e.g., 1 wei) of the `newToken`. This inserts `newToken` into the rewards array prematurely. When the admin's transaction executes afterward, it appends `newToken` again, resulting in duplicate entries in the rewards array.

— RECOMMENDATION

Add a dedicated function to explicitly deactivate a reward token, which ensures a clean removal from the rewards array.

— RESOLUTION

Status: Pending.

LOW **L-36**

PENDING

Missing ERC20 Recover Functions in Contracts

CATEGORY Informational

LOCATION Global

REVIEW Main Review

— DESCRIPTION

The Voter, Gauge, and Bribe contracts are designed to hold reward tokens, which users can claim through their respective claim functions. However, there is no `recoverERC20` function that allows the admin to retrieve leftover tokens from these contracts. This limitation means that old or deprecated reward tokens, as well as small dust amounts that accumulate due to rounding, may become permanently stuck in the contracts.

— RECOMMENDATION

Add a `recoverERC20` function to each of these contracts, restricted to the admin, to allow the retrieval of stranded tokens.

— RESOLUTION

Status: Pending.

LOW **L-37**

PENDING

Wrong maxRuns usage

CATEGORY Best Practices

LOCATION Gauge.sol

REVIEW Main Review

— DESCRIPTION

The Gauge functions `_batchRewardPerToken` and `_updateRewardPerToken()` accept a `maxRuns` parameter which should be used to determine how much iterations should be performed at most. However, it's used incorrectly like this

```
``plain text
uint256 _startIndex = getPriorSupplyIndex(_startTimestamp);
uint256 _endIndex = Math.min(supplyNumCheckpoints - 1, maxRuns);
`
```

If `maxRuns = 2`, the code will loop until index 2, not 2 times. Which means that if the `startIndex` is greater than `maxRuns` it won't loop at all.

— RECOMMENDATION

Consider either changing the name of the parameter to `maxIndex` or change the logic to count the iterations.

— RESOLUTION

Status: Pending.

LOW **L-38**

PENDING

Some voter functions can be DOSed

CATEGORY	Informational
LOCATION	Voter.sol
REVIEW	Main Review

— DESCRIPTION

Creating gauges is a permissionless operation, the only two conditions that must be met are:

- The pool the gauge is associated with must be created through the factory
- The pool's tokens must be whitelisted

A Malicious user can create an infinite amount of pools by tweaking the configuration values and then use them to create a gauge for each of them in the `Voter` contract. This will cause a huge growth in the size of `allGauges` and can effectively DOS all functions that rely on it - `updateAll()`, `distroFees()`, `distro()`, `distribute()` and also all functions using `Voter.length()`.

— RECOMMENDATION

Acknowledge the risk.

— RESOLUTION

Status: Pending.

LOW **L-39**

PENDING

Bribe.left() is not necessary

CATEGORY	Informational
LOCATION	Bribe.sol, Gauge.sol
REVIEW	Main Review

— DESCRIPTION

Currently, `Bribe.left()` is used by the `Gauge` to send the accrued fees if they are greater than the returned value. This approach is not needed in the `Bribe` because rewards are added towards `tokenRewardsPerEpoch`, so even if the amount notified is less than that, it would be correctly assigned.

— RECOMMENDATION

Consider removing the `left()` function from the `Bribe` and its usage in the `Gauge`.

— RESOLUTION

Status: Pending.

§ 05

Disclaimer.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian's position is that each company and individual are responsible for their own due diligence and continuous security. Guardian's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.