platform (Convex Finance)

Benchmark Security Review Performed by Bug Hunter

17.05.2025

Table of Contents

About Bug Hunter

Bug Hunter is an automated code-review tool for Solidity smart contracts, developed by Truscova GmbH. It uses proprietary algorithms and advanced code-analysis techniques to identify potential vulnerabilities efficiently.

Bug Hunter is designed to support developers in the Ethereum ecosystem by delivering consistent, scalable assessments that align with best practices in Web 3.0 security. Reports are generated via the Bug Hunter tool and are accessible to clients through a secure dashboard web interface or relevant Git repositories.

Bug Hunter is a product of Truscova, a security R&D company headquartered in Bremen, Germany. Founded in late 2022, Truscova specializes in formal verification, fuzzing, dynamic and static analysis, and security tooling. Its founding team includes researchers with 35+ books, several patents, and over 20 000 citations in software testing, verification, security, and machine learning.

To explore more about Bug Hunter, visit:

- Bug Hunter Website
- Bug Hunter Documentation
- Example Reports

To explore more about Truscova, visit:

- Truscova Website
- X
- LinkedIn

Notices and Remarks

Copyright and Distribution

This security-review report was generated with Truscova's Bug Hunter tool to benchmark its performance on selected Yield Farming projects. It is made available through the Bug Hunter dashboard and relevant Git repositories. Users may choose to share or publish this report at their own discretion.

Test Coverage Disclaimer

This security review was conducted exclusively by Bug Hunter, an automated code-review tool for Solidity codebases. Bug Hunter uses a combination of proprietary algorithms and programmatic techniques to identify potential vulnerabilities in Solidity smart contracts.

The findings in this report should not be considered a complete or exhaustive list of all possible security issues in the reviewed codebase and are dependent on Bug Hunter's coverage of detector rules. The full list of rules covered by Bug Hunter automated code review is available at https://docs.bughunter.live/.

Executive Summary

Overview and Scope

A codebase from project platform was reviewed for security vulnerabilities by Bug Hunter with the following details:

- GitHub Repository: https://github.com/convex-eth/platform
  Commit hash: `754d9e700693246275b613e895b4044b63ce9ed5`

Bug Hunter reviewed the following files:

VoterProxy.sol BaseRewardPool.sol CrvDepositor.sol Interfaces.sol StashFactory.sol DepositToken.sol Cvx.sol ExtraRewardStashV2.sol Booster.sol ManagedRewardPool.sol RewardFactory.sol cCrv.sol DebugInterfaces.sol cCrvRewardPool.sol TokenFactory.sol ExtraRewardStashV1.sol cvxRewardPool.sol VirtualBalanceRewardPool.sol

Summary of Findings

The uncovered vulnerabilities identified during the security review are summarised in the table below:

Severity	Count
High	0
Medium	2
Low	3

Project Summary

Contact Information

For support, contact us at support@bughunter.live

Project Timeline

Date	Event
17.05.2025	Project submitted
17.05.2025	Security review concluded

Detailed Findings

1. Missing Null Address Check for `_gauge` Parameter

Severity	Medium
Finding ID	BH-platform-01
Target	contracts/contracts/Booster.sol
Function name	addPool

Description

The `addPool` function does not validate that the `_gauge` address provided as input is non-zero. This could potentially allow the addition of a pool with an invalid gauge address (e.g., the zero address), leading to unexpected behavior, misconfiguration, or making the pool unusable. While the function does check if `_gauge` exists in the gauge list from the registry, it still makes sense to explicitly reject null addresses to avoid subtle bugs or exploit vectors if external contracts behave unexpectedly.

```
function addPool(address _swap, address _gauge, uint256 _stashVersion) external {
    require(_gauge != address(0), "invalid gauge address");

    // get curve's registry
    address mainReg = IRegistry(registry).get_registry();

    // get lp token and gauge list from swap address
    address lptoken = IRegistry(mainReg).get_lp_token(_swap);
    (address[10] memory gaugeList, ) = IRegistry(mainReg).get_gauges(_swap);

    // confirm the gauge passed in calldata is in the list
    bool found = false;
    for (uint256 i = 0; i < gaugeList.length; i++) {
        if (gaugeList[i] == _gauge) {
            found = true;
            break;
        }
    }
    require(found, "!registry");

    // ensure this pool/gauge hasn't been added before
    found = false;
    for (uint256 i = 0; i < poolInfo.length; i++) {
        if (poolInfo[i].gauge == _gauge) {
            found = true;
            break;
        }
    }
    require(!found, "already registered");

    // the next pool's pid
    uint256 pid = poolInfo.length;

    // create a tokenized deposit
    address token = ITokenFactory(tokenFactory).CreateDepositToken(lptoken);
    // create a reward contract for crv rewards
    address newRewardPool = IRewardFactory(rewardFactory).CreateCrvRewards(pid,
token);
    // create a stash to handle extra incentives
    address stash = IStashFactory(stashFactory).CreateStash(pid, _gauge, staker,
_stashVersion);

    // add the new pool
    poolInfo.push(
        PoolInfo({
            lptoken: lptoken,
            token: token,
            gauge: _gauge,
            crvRewards: newRewardPool,
            stash: stash
        })
    );

    // give stashes access to rewardfactory and voteproxy
    if (stash != address(0)) {
        poolInfo[pid].stash = stash;
        IStaker(staker).setStashAccess(stash, true);
        IRewardFactory(rewardFactory).setAccess(stash, true);
    }
}
```

Recommendation

Add a validation step at the beginning of the function to ensure that `_gauge` is not the zero address:

```
require(_gauge != address(0), "invalid gauge address");
```

2. Arbitrary External Call in `vote` Function

Severity	Low
Finding ID	BH-platform-02
Target	contracts/contracts/Booster.sol
Function name	vote

Description

The `vote` function allows the `operator` to call any contract by passing its address through the `_votingAddress` parameter. Since no checks are performed on `_votingAddress`, this creates a powerful primitive for interacting with arbitrary contracts. If the `operator` role is compromised or misused, this could lead to arbitrary contract calls, potentially resulting in loss of funds or unintended behavior.

```
function vote(uint256 _voteId, address _votingAddress, bool _support) external
returns (bool) {
    require(msg.sender == voteDelegate, "!auth");
    require(approvedVotingContracts[_votingAddress], "unauthorized voting address");
    IStaker(staker).vote(_voteId, _votingAddress, _support);
    return true;
}
```

Recommendation

Implement a whitelist of allowed voting contract addresses or validate that `_votingAddress` is a trusted, expected contract. For example:

```
require(approvedVotingContracts[_votingAddress], "unauthorized voting address");
```

Also ensure the `operator` role is tightly controlled.

3. Front-Running Risk in `setFactories` Function

Severity	Low
Finding ID	BH-platform-03
Target	contracts/contracts/Booster.sol
Function name	setFactories

Description

The `setFactories` function allows the `owner` to change the factory addresses without any delay or safeguards. This introduces a front-running risk: an attacker monitoring the mempool could observe an `addPool` transaction relying on current factory addresses, and if the `owner` sends a `setFactories` transaction with malicious addresses in the same block or just before, they could potentially divert reward or token creation logic to malicious contracts. This creates a window of opportunity for exploit or misbehavior if the `owner` key is compromised or misused.

```
uint256 public factoryUpdateDelay = 1 days;
uint256 public factoryUpdateTimestamp;
address public pendingRewardFactory;
address public pendingStashFactory;
address public pendingTokenFactory;

function setFactories(address _rfactory, address _sfactory, address _tfactory)
external {
    require(msg.sender == owner, "!auth");

    // Queue the update with a delay
    pendingRewardFactory = _rfactory;
    pendingStashFactory = _sfactory;
    pendingTokenFactory = _tfactory;
    factoryUpdateTimestamp = block.timestamp;
}

function executeFactoryUpdate() external {
    require(msg.sender == owner, "!auth");
    require(block.timestamp >= factoryUpdateTimestamp + factoryUpdateDelay, "update
not yet executable");
```

```
    rewardFactory = pendingRewardFactory;
    stashFactory = pendingStashFactory;
    tokenFactory = pendingTokenFactory;
}
```

Recommendation

Implement a time-delay mechanism (e.g., timelock) for sensitive operations like updating factory addresses. Alternatively, restrict factory address updates to a one-time initialization period or enforce multi-signature approval for such changes. Example:

```
require(pendingUpdate.timestamp + delay < block.timestamp, "update not yet
executable");
```

4. Missing `safeApprove` Usage Before `approve` Call

Severity	Low
Finding ID	BH-platform-04
Target	contracts/contracts/Booster.sol
Function name	deposit

Description

In the `deposit` function, the contract calls `IERC20(token).approve(rewardContract, _amount)` without resetting the approval to zero first. This can lead to compatibility issues with certain ERC20 tokens that require allowance to be set to zero before a new non-zero value is approved. Additionally, the code uses `approve` directly rather than `safeApprove`, which can lead to unexpected failures or unsafe behavior with non-standard ERC20 implementations.

```
function deposit(uint256 _pid, uint256 _amount, bool _stake) public returns (bool) {
    require(!isShutdown, "shutdown");

    address lptoken = poolInfo[_pid].lptoken;
    IERC20(lptoken).safeTransferFrom(msg.sender, address(this), _amount);

    // move to curve gauge
    sendTokensToGauge(_pid);

    address token = poolInfo[_pid].token;
    if (_stake) {
        // mint here and send to rewards on user's behalf
        ITokenMinter(token).mint(address(this), _amount);
        address rewardContract = poolInfo[_pid].crvRewards;

        // safely reset and set allowance
        IERC20(token).safeApprove(rewardContract, 0);
        IERC20(token).safeApprove(rewardContract, _amount);

        IRewards(rewardContract).stakeFor(msg.sender, _amount);
    } else {
        // add user balance directly
        ITokenMinter(token).mint(msg.sender, _amount);
    }
    emit Deposited(msg.sender, _pid, _amount);
    return true;
}
```

Recommendation

Use `safeApprove` from OpenZeppelin's `SafeERC20` library instead of `approve`. Also consider resetting allowance to zero before setting a new value to ensure compatibility:

```
IERC20(token).safeApprove(rewardContract, 0);
IERC20(token).safeApprove(rewardContract, _amount);
```

5. `withdrawAll` Ignores Local Token Balance, Risking Lost Funds

Severity	Medium
Finding ID	BH-platform-05
Target	contracts/contracts/VoterProxy.sol
Function name	withdrawAll

Description

The `withdrawAll` function calculates the withdrawal amount using only `balanceOfPool(_gauge)`, which returns the amount staked in the gauge. It does not account for any `_token` balance that may already be held by the `VoterProxy` contract. As a result, if tokens were sent directly to the contract or previously withdrawn but not yet transferred, they would be excluded from the withdrawal logic. This could lead to stranded funds, particularly during a system shutdown or emergency migration scenario.

```
function withdrawAll(address _token, address _gauge) external returns (bool) {
    require(msg.sender == operator, "!auth");

    uint256 staked = balanceOfPool(_gauge);
    uint256 local = IERC20(_token).balanceOf(address(this));
    uint256 total = staked + local;

    withdraw(_token, _gauge, total);
    return true;
}
```

Recommendation

Modify `withdrawAll` to include both the staked tokens (via `balanceOfPool`) and the local token balance (`IERC20(_token).balanceOf(address(this))`) to ensure the entire token balance is withdrawn. For example:

```
uint256 staked = balanceOfPool(_gauge);
uint256 local = IERC20(_token).balanceOf(address(this));
withdraw(_token, _gauge, staked + local);
```