e

CD SECURITY

AUDIT REPORT

PowerX
December 2024

Prepared by
GT_GSEC
MrPotatoMagic

Introduction

A time-boxed security review of the **PowerX** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **PowerX**

PowerX is a referral system for that enables multi-level marketing (MLM) structures. With a flexible commission structure participants are paid using both ERC20 and ETH tokens.

The first product using the referral system are the Wrapped Hydra Miners (WHydra) which represent time-bound mining operations. The protocol handles minting and claiming WHydra tokens as well as the automatic distribution of the commission generated through the user's payments to referrers and partners.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 4924e736a2863e0ab3a8984f52f8e805d0775c1e**

Scope

The following smart contracts were in scope of the audit:

- contracts/*

The following number of issues were found, categorized by their severity:

- Medium: 1 issues
- Low: 2 issues
- Info: 11 issues

Findings Summary

ID	Title	Severity	Status
[M-01]	Incorrect TWAP quote price retrieval due to usage of same TWAP `secondsAgo` for all pools in `REFUND_WETH_PATH`	Medium	Acknowledged
[L-01]	`WHydra` does not benefit from the `_safeMint` function	Low	Fixed
[L-02]	Slippage and TWAP can be under third party control within the `claimMint` function	Low	Acknowledged
[I-01]	Missing tree update in function `moveUser()`	Info	Fixed
[I-02]	Consider ensuring user `!=` to in function `moveUser()`	Info	Fixed
[I-03]	The `mintPower` parameter has incomplete input validation	Info	Fixed
[I-04]	The `startMint` can revert with unhandled integer underflow error	Info	Fixed
[I-05]	Check Effects Interaction pattern violation	Info	Fixed
[I-06]	The `earlyEndMint` function does not check minimum maturity days	Info	Fixed
[I-07]	The `_handleCommission` function makes use of `IERC20`'s `approve` function	Info	Fixed
[I-08]	Lack of event emission on state change	Info	Acknowledged
[I-09]	The `NON_FUNGIBLE_POSITION_MANAGER` constant is unused	Info	Acknowledged
[I-10]	The `Swap.buy` is used only with the native token	Info	Acknowledged
[I-11]	Essential configuration parameters lack input validation within `Referral`	Info	Acknowledged

Detailed Findings

[M-01] Incorrect TWAP quote price retrieval due to usage of same TWAP `secondsAgo` for all pools in

REFUND_WETH_PATH

Severity

Impact: Medium

Likelihood: Medium

Description

Function estimateMaximumInputAmount() and estimateMinimumOutputAmount() call function getQuote() in the for loop. We notice that when there are multiple hops, we utilize the same TWAP period for all pools. This is true for the REFUND_WETH_PATH case in WHydra.sol when refunds occur in claimMint().

Due to this, it is possible for the final quote amount to deviate and provide the user with a bad refund price.

```
  function estimateMaximumInputAmount(SwapParams memory params) public
view returns (uint256) {
    PathDecoder.Hop[] memory hops = PathDecoder.decode(params.path);
    uint256 amount = params.amount;

    for (uint256 i = hops.length; i > 0; i--) {
      PathDecoder.Hop memory hop = hops[i - 1];
      (amount, ) = getQuote(hop.tokenOut, hop.tokenIn, hop.fee,
params.twap, amount);
    }

    return (amount * (Constants.BASIS + params.slippage)) /
Constants.BASIS;
  }

  function estimateMinimumOutputAmount(SwapParams memory params) public
view returns (uint256) {
    PathDecoder.Hop[] memory hops = PathDecoder.decode(params.path);
    uint256 amount = params.amount;

    for (uint256 i = 0; i < hops.length; i++) {
      (amount, ) = getQuote(hops[i].tokenIn, hops[i].tokenOut,
hops[i].fee, params.twap, amount);
    }

    return (amount * (Constants.BASIS - params.slippage)) /
Constants.BASIS;
  }
```

Recommendations

For refunding through WHydra.sol, consider passing an array of twap secondsAgos. These values can be kept configurable by only the team to ensure the TWAP period is neither too long nor too short.

[L-01] `WHydra` does not benefit from the `_safeMint` function

Description

The `WHydra` is an ERC712 token implementation. Within the `_startMint` function it mints a Hydra tokens purchaser a NFT by means of the `_mint` function. Thus, it does not benefit from the `ERC721TokenReceiver` interface that can be implemented by the NFT receiver. This interface implementation ensures that recipient is aware of NFT receiving, so no token will stuck within the contract.

```
contract WHydra is ERC721Enumerable, AccessManaged {
...
  function _startMint(address to, uint256 mintPower, uint256 numOfDays)
private returns (Token storage token) {
    // Start the mint on the active proxy
    uint256 id = activeProxy.startMint(mintPower, numOfDays);

    // Mint the NFT
    uint256 tokenId = nextTokenId;
    nextTokenId++;

    // Assign mint details
    token = tokens[tokenId];
    token.mint.proxy = address(activeProxy);
    token.mint.id = id;

    // Mint the NFT to the recipient
    _mint(to, tokenId);

    // Emit Event
    emit StartMint(msg.sender, to, tokenId);

    return token;
  }
  }
```

Recommendations

It is recommended to use `_safeMint` instead of `_mint` function while minting new tokens.

[L-02] Slippage and TWAP can be under third party control within the `claimMint` function

Description

The `claimMint` function can be called by both NFT owner or by the `trusted` address. The `trusted` is a collection of bots' addresses that can claim `Hydra` tokens on behalf of the users. However, whenever the trusted address claims the mint, it provides the `slippage` and `twap` input parameters on its own. This brings a risk that slippage percentage value might be too generous and the TWAP period too long or too short. As a result, the custom, third party configuration may have negative impact on the final swap exchange rate done before the native token refund functionality.

```
  function claimMint(ClaimMintParams calldata params) external { //@audit
trusted controls slippage
    Token storage token = tokens[params.tokenId];

    if (params.slippage > maxSlippage) revert InvalidSlippage();
    if (token.mint.id == 0) revert InvalidTokenId();

    // Ensure only the NFT owner or a trusted source can perform the claim
    address nftOwner = _ownerOf(params.tokenId);
    if (nftOwner != msg.sender && !trusted[msg.sender]) revert
InvalidCaller();
...

    // Process refunds if enabled
    if (token.doRefund && token.mintPayment.asset == Constants.WETH) {
      token.mintRefund.refund.asset = Constants.WETH;
      token.mintRefund.spent.asset = HYDRA;

      Swap.SwapResult memory result = Swap.refund(
        Swap.SwapParams({
          path: REFUND_WETH_PATH,
          slippage: params.slippage,
          twap: params.twap,
          amount: token.mintPayment.amount + token.mintCommission.amount,
          recipient: nftOwner,
          deadline: params.deadline
        })
      );

      token.mintRefund.refund.amount = result.received;
      token.mintRefund.spent.amount = result.spent;
    }
...
  }
```

Recommendations

It is recommended to review the design and defined business rules within the protocol. The mitigation technique might be dependant on the several factors, and can be extended by among the others:

- Allowing user to configure and save slippage and TWAP period in advance. Then a bot can fetch the saved config upon calling the `claimMint`. function
- Documenting clearly the behavior so the end user is aware of the possible risk.

[I-01] Missing tree update in function `moveUser()`

Description

In function moveUser(), we do not call `set._tree` to update the referrerId.

```
  function moveUser(Set storage set, string memory user, string memory to)
internal {
    if (!isValidId(set, user) || !isValidId(set, to)) {
      revert InvalidId();
    }

    Info storage userInfo = _getInfoFromId(set, user.toShortString());
    ShortString from = userInfo.onboardedBy;

    userInfo.onboardedBy = to.toShortString();
    emit UserMoved(user.toShortString(), from, to.toShortString());
  }
```

This is correctly done in the onboardAndAddCommission() function but not in function moveUser(). The tree does not seem to be used anywhere specifically, due to which the severity of this issue is Low.

```
// Onboard users only when initially created
      if (created) {
        userInfo.onboardedBy = referrerId;
        set._tree[userInfo.id] = referrerId;

        emit OnboardedBy(userInfo.id, referrerId);
      }
```

Recommendations

As done during onboarding, update the `set._tree` value for the id in function moveUser().

[I-02] Consider ensuring user `!=` to in function `moveUser()`

Description

In function moveUser(), we do not ensure that the `user` and `to` parameter are not the same. While moveUser() is restricted in PowerX.sol, the check should be implement to safeguard against self referrals.

```
function moveUser(Set storage set, string memory user, string memory to)
internal {
```

```
    if (!isValidId(set, user) || !isValidId(set, to)) {
      revert InvalidId();
    }

    Info storage userInfo = _getInfoFromId(set, user.toShortString());
    ShortString from = userInfo.onboardedBy;

    userInfo.onboardedBy = to.toShortString();
    emit UserMoved(user.toShortString(), from, to.toShortString());
  }
```

Recommendations

Check `to` to not be the `user`.

[I-03] The `mintPower` parameter has incomplete input validation

Description

The `startMint` function does input validation for the `mintPower` parameter. However, it only checks whether the value is potentially too high.

```
  function startMint(StartMintParams calldata params) external payable
returns (Token memory) {
    if (params.slippage > maxSlippage) revert InvalidSlippage();
    if (params.mintPower > MAX_MINT_POWER_CAP) revert
MintPowerOutOfBound();
    ...
```

The `Hydra` token's `_startMint` function also checks whether the `mintPower` is not equal to 0. Thus, the `WHydra` lacks part of validation which may lead to transaction revert in the late processing.

```
    function _startMint(
        address user,
        uint256 mintPower,
        uint256 numOfDays,
        uint256 mintableHydra,
        uint256 gMintPower,
        uint256 currentHRank,
        uint256 mintCost
    ) internal returns (uint256 mintable) {
        if (numOfDays == 0 || numOfDays > MAX_MINT_LENGTH) revert
Hydra_InvalidMintLength();
        if (mintPower == 0 || mintPower > MAX_MINT_POWER_CAP) revert
Hydra_InvalidMintPower();
```

The analogical scenario applies for `numOfDays` as well.

Recommendations

It is recommended to consider implementing mirrored input validation for the `mintPower` parameter in `WHydra` as it is done in `Hydra` to prevent transaction late reverting.

[I-04] The `startMint` can revert with unhandled integer underflow error

Description

The `startMint` function allows user to start `Hydra` tokens minting by paying in `TitanX` or in native token. Whenever the Ether is used, the function firstly swaps an amount for `TitanX` tokens. Then, it calculates the commission that must be paid as a form of fee. Finally, it checks whenever a refund of `remaining` amount should be transferred back to the purchaser.

```
...
      uint256 remaining = msg.value - commission - result.spent;
      if (remaining > 0) {
        Address.sendValue(payable(msg.sender), remaining);
      }
...
```

However, the function does not check whether the Ether provided by the user (`msg.value`) is higher than Ether spent for swap and required for commission. Thus, in the event of insufficient amount of Ether is provided the transaction can revert with unhandled integer underflow error.

```
  function startMint(StartMintParams calldata params) external payable
returns (Token memory) {
    if (params.slippage > maxSlippage) revert InvalidSlippage();
    if (params.mintPower > MAX_MINT_POWER_CAP) revert
MintPowerOutOfBound();

    // Ensure valid referrer if minting is closed
    if (!powerX.isValidId(params.referrer)) revert InvalidReferrer();

    if (closedMint) {
      if (powerX.isNullId(params.referrer)) revert InvalidReferrer();
    }

    // Deploy a new proxy if necessary
    _deployNewProxyIfNecessary();

    // Calculate mint cost
    uint256 mintCost = _mintCost(params.mintPower);
```

```
    uint256 commission;
    uint256 payment;
    ShortString onboardedBy;
    ShortString userId;

    if (params.payWithETH) {
      // Swap ETH to TitanX
      Swap.SwapResult memory result = Swap.buy(
        Swap.SwapParams({
          path: BUY_TITANX_PATH,
          slippage: params.slippage,
          twap: params.twap,
          amount: mintCost,
          recipient: address(activeProxy),
          deadline: params.deadline
        }),
        true
      );

      // Track spendings in WETH
      payment = result.spent;

      // Calculate commission
      (commission, userId, onboardedBy) = _handleCommission(
        HandleCommissionParams({
          asset: address(0),
          amount: result.spent,
          user: params.to,
          userId: '',
          referrerId: params.referrer,
          onboard: true
        })
      );

      // Refund remaining ETH
      uint256 remaining = msg.value - commission - result.spent;
      if (remaining > 0) {
        Address.sendValue(payable(msg.sender), remaining); //@audit
reentrancy possibility. CEI violation
      }
    } else {
      // Calculate TitanX commission
```

Recommendations

It is recommended to consider implementing a verification whether a sufficient amount of Ether was added to the transaction and handle a possible error.

[I-05] Check Effects Interaction pattern violation

Description

The `startMint` and `earlyEndMint` functions do not follow the Check-Effect-Interaction (CEI) pattern flawlessly. Within middle of the functions they can transfer back a refund in native coin to the purchaser. This action can trigger the recipient's `fallback` or `receive` function and allow reenter the contract. Afterwards various state changes occur, e.g. update of the `Token` storage structure. However, no exploitation path were identified within the time of the security assessment. The finding is reported as a deviation from leading security practices.

```
  function startMint(StartMintParams calldata params) external payable
returns (Token memory) {
...
    if (params.payWithETH) {
...
      // Refund remaining ETH
      uint256 remaining = msg.value - commission - result.spent;
      if (remaining > 0) {
        Address.sendValue(payable(msg.sender), remaining); //@audit
reentrancy possibility. CEI violation
      }
    } else {
      // Calculate TitanX commission
...
    }

    // Start the mint
    Token storage token = _startMint(params.to, params.mintPower,
params.numOfDays);

    // Update token details
    token.mint.referrer = onboardedBy;
    token.mint.mintedTo = userId;
    token.doRefund = params.doRefund;

    token.mintCommission.asset = params.payWithETH ? Constants.WETH :
TITANX;
    token.mintCommission.amount = commission;

    token.mintPayment.asset = params.payWithETH ? Constants.WETH : TITANX;
    token.mintPayment.amount = payment;

    return token;
}
```

Recommendations

It is recommended to follow the CEI pattern in every case. In the case of the aforementioned functions, the refund should occur after all state changes are applied.

[I-06] The `earlyEndMint` function does not check minimum maturity days

Description

The `earlyEndMint` function allows to end the `Hydra` tokens mint earlier by providing additional payment of the `TitanX` tokens. However, the `Hydra` token's `_earlyEndMint` function requires that minimum 3 days passed since the mint period started. Thus, the `WHydra` lacks part of the input validation which may lead to transaction revert in the late processing.

```
    function _earlyEndMint(
        address user,
        uint256 id
    ) internal returns (uint256 reward, uint256 mintPower) {
...
        //revert if miner has matured or less than 3 days
        uint256 daysMatured = (block.timestamp - mint.mintStartTs) / 1
days;
        if (mint.maturityTs <= block.timestamp || daysMatured < 3)
            revert Hydra_MintMaturityNotMet();
...
```

Recommendations

It is recommended to consider implementing mirrored validation check of the days passed since the mint period started in `WHydra` as it is done in `Hydra` to prevent transaction late reverting.

[I-07] The `_handleCommission` function makes use of `IERC20`'s `approve` function

Description

The `_handleCommission` makes use of the standard IERC20's `approve` function to set the allowance for the `PowerX` contract. The usage of the `approve` is discouraged as it may lead to security issues whenever the calling contract does not consume all given allowance. However, no such instance is identified for `Hydra` or `TitanX` tokens. Additionally, in other instances the protocol makes use of the `SafeERC20`'s `safeIncreaseAllowance` function to set desired allowance.

```
function _handleCommission(
    HandleCommissionParams memory params
  )
    private
    returns (
      uint256 commission,
      ShortString userOnboardId, // Only relevant for onboarding
      ShortString onboardedBy // Only relevant for onboarding
    )
  {
```

```
...
      // Ensure sufficient approval //@audit commission would be
sufficient
      token.approve(address(powerX), token.balanceOf(address(this)));
//@audit approve is not considered secure
...
```

Recommendations

It is recommended to consider using the `SafeERC20`'s `safeIncreaseAllowance` or `forceApprove` functions instead of `approve`.

[I-08] Lack of event emission on state change

Description

Few functions such as `setClosedMint`, `setMaxSlippage` and `setTrusted` lack of event emission upon successful key configuration update. Such omission may lead to hindered off-chain monitoring or investigation in the event of the security breach.

```
  /**
   * @notice Toggles whether minting is closed
   * @param closedMint_ New closed mint status
   */
  function setClosedMint(bool closedMint_) external restricted {
    closedMint = closedMint_;
  }

  /**
   * @notice Set the maximum slippage acceptable on swaps
   * @param maxSlippage_ the maximum acceptable slippage
   */
  function setMaxSlippage(uint256 maxSlippage_) external restricted {
    // Limit maximum slippage to 25%
    if (maxSlippage_ > 2500) revert InvalidSlippage();
    maxSlippage = maxSlippage_;
  }

  /**
   * @notice Set trusted sources
   * @param sender the msg.sender address
   * @param isTrusted true if trusted, false otherwise
   */
  function setTrusted(address sender, bool isTrusted) external restricted
{
    trusted[sender] = isTrusted;
  }
```

Recommendations

It is recommended to include event emissions within the key functionalities to improve the off-chain monitoring possibilities.

[I-09] The `NON_FUNGIBLE_POSITION_MANAGER` constant is unused

Description

The `NON_FUNGIBLE_POSITION_MANAGER` constant is defined but never used. Thus, it impacts on the deployment Gas cost and the size of the contract's bytecode.

```
  /**
   * @dev UniSwap V3 Non-Fungible Position Manager address
   */
  address constant NON_FUNGIBLE_POSITION_MANAGER =
0xC36442b4a4522E871399CD717aBDD847Ab11FE88;
```

Recommendations

It is recommended to remove unused constants to reduce binary size and deployment Gas cost.

[I-10] The `Swap.buy` is used only with the native token

Description

The `buy` function from the `Swap` library allows to swap either native token for some token or simply swap ERC20 tokens. The `useEth` input flag controls which flow is executed. However, it was identified that this function is called only with `useEth` set to true within the protocol. Thus, the second path of execution is never used. As a result, unused code increases only the library size and deployment Gas cost.

```
function buy(SwapParams memory params, bool useEth) external returns
(SwapResult memory result) {
    PathDecoder.Hop[] memory hops = PathDecoder.decode(params.path);
    IERC20 paymentToken = IERC20(hops[0].tokenIn);
    uint256 amountInMax = estimateMaximumInputAmount(params);
    bytes memory reversedPath = PathDecoder.encodeReverse(hops);

    bytes[] memory inputs = new bytes[](useEth ? 3 : 1);
    bytes memory commands;

    result.received = params.amount;
```

```
    if (useEth) {
..
    } else {
      commands = abi.encodePacked(SWAP_EXACT_OUT);
      _approveForSwap(paymentToken, amountInMax);
      inputs[0] = abi.encode(params.recipient, params.amount, amountInMax,
reversedPath, SOURCE_MSG_SENDER);

      uint256 balanceBefore = paymentToken.balanceOf(address(this));
      IUniversalRouter(Constants.UNIVERSAL_ROUTER).execute(commands,
inputs, params.deadline);
      result.spent = balanceBefore -
paymentToken.balanceOf(address(this));
    }
  }
```

Recommendations

It is recommended to consider removal of unused code to save some Gas.

[I-11] Essential configuration parameters lack input validation within `Referral`

Description

The `_setPartnerAllocation`, `_setPercentage`, `setLevels` functions from the `Referral` library lacks any kind of input validation for the key parameters: `allocation`, `levels`, `percentage` . Thus, the protocol owner can set any value for aforementioned parameters, e.g. an allocation representing the percentage of commission applicable, including even a value exceeding 100%.

```
  /**
   * @notice Private helper to set the partner allocation for a given user
info.
   * @param set The referral set to operate on.
   * @param caller The address of the caller setting the allocation.
   * @param asset The asset to allocate.
   * @param info The user info struct.
   * @param allocation The allocation percentage to set.
   */
  function _setPartnerAllocation(
    Set storage set,
    address caller,
    address asset,
    Info storage info,
    uint256 allocation
  ) private {
    info.partnerAllocations[caller].set(asset, allocation);

    bytes32 partnerId = ShortString.unwrap(info.id);
```

```
    // Add or remove partnerId based on existence of any allocations
    if (info.partnerAllocations[caller].any()) {
      set._partners[caller].add(partnerId);
    } else {
      set._partners[caller].remove(partnerId);
    }

    emit PartnerAllocationSet(caller, asset, info.id, allocation);
  }
  ...
    function setLevels(Set storage set, address caller, address asset,
uint256[] memory v) internal {
    set._levels[caller][asset] = v;

    emit LevelsSet(caller, asset, v);
  }
  ...
    function _setPercentage(Info storage info, address caller, address
asset, uint256 percentage) private {
    info.overrides[caller].set(asset, percentage);
    emit PercentageOverrideSet(caller, asset, info.id, percentage);
  }
```

Recommendations

It is recommended to consider implementing input validation for the aforementioned functionality.