



HighStreetMarket – ProductToken

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: December 27th, 2021 – December 31st, 2021

Visit: Halborn.com

DOCUMENT REVISION HISTORY	3
CONTACTS	3
1 EXECUTIVE OVERVIEW	4
1.1 INTRODUCTION	5
1.2 AUDIT SUMMARY	5
1.3 TEST APPROACH & METHODOLOGY	5
RISK METHODOLOGY	6
1.4 SCOPE	8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	9
3 FINDINGS & TECH DETAILS	10
3.1 (HAL-01) PRODUCTTOKENHIGHBASE CONTRACT IS VULNERABLE TO SANDWICH ATTACKS - HIGH	12
Description	12
Proof of Concept	14
Risk Level	15
Recommendation	16
Remediation Plan	16
3.2 (HAL-02) MISSING ZERO ADDRESS CHECK - LOW	17
Description	17
Code location	17
Risk Level	18
Recommendation	18
Remediation Plan	18
3.3 (HAL-03) EXTERNAL CALLS WITHIN A LOOP - LOW	19
Description	19

Proof of Concept	21
Risk Level	21
Recommendation	22
Remediation Plan	22
3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL	
23	
Description	23
Risk Level	23
Recommendation	23
Remediation Plan	23
3.5 (HAL-05) TYPO IN FUNCTION NAMES AND EVENTS - INFORMATIONAL	24
Description	24
Risk Level	24
Recommendation	25
Remediation Plan	25
4 AUTOMATED TESTING	26
4.1 STATIC ANALYSIS REPORT	27
Description	27
Slither results	27
4.2 AUTOMATED SECURITY SCAN	35
Description	35
MythX results	35

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/27/2021	Roberto Reigada
0.2	Document Updates	12/31/2021	Roberto Reigada
0.3	Draft Review	01/03/2022	Gabi Urrutia
1.0	Remediation Plan	01/10/2021	Roberto Reigada
1.1	Remediation Plan Review	01/10/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

HighStreetMarket engaged Halborn to conduct a security audit on their smart contracts beginning on December 27th, 2021 and ending on December 31st, 2021. The security assessment was scoped to the smart contracts provided in the Github repository [Highstreet-World/ProductToken](#)

1.2 AUDIT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were addressed by the [HighStreetMarket team](#).

1.3 TEST APPROACH & 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 audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- 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
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. ([MythX](#))
- Static Analysis of security for scoped contract, and imported functions. ([Slither](#))
- Testnet deployment ([Brownie](#), [Remix IDE](#))

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of **5 to 1** with **5** being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

- 5 - Almost certain an incident will occur.
- 4 - High probability of an incident occurring.
- 3 - Potential of a security incident in the long term.
- 2 - Low probability of an incident occurring.
- 1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

- 5 - May cause devastating and unrecoverable impact or loss.
- 4 - May cause a significant level of impact or loss.

EXECUTIVE OVERVIEW

- 3 - May cause a partial impact or loss to many.
- 2 - May cause temporary impact or loss.
- 1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of **10** to **1** with **10** being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
----------	------	--------	-----	---------------

- 10** - CRITICAL
- 9** - **8** - HIGH
- 7** - **6** - MEDIUM
- 5** - **4** - LOW
- 3** - **1** - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contracts:

- BancorBondingCurve.sol
- Power.sol
- ProductNft.sol
- ProductTokenCore.sol
- ProductTokenHighBase.sol

Commit ID: [9be0879308ad3e4fa43a8793eb5e580f60e291b1](#)

Fixed Commit ID:

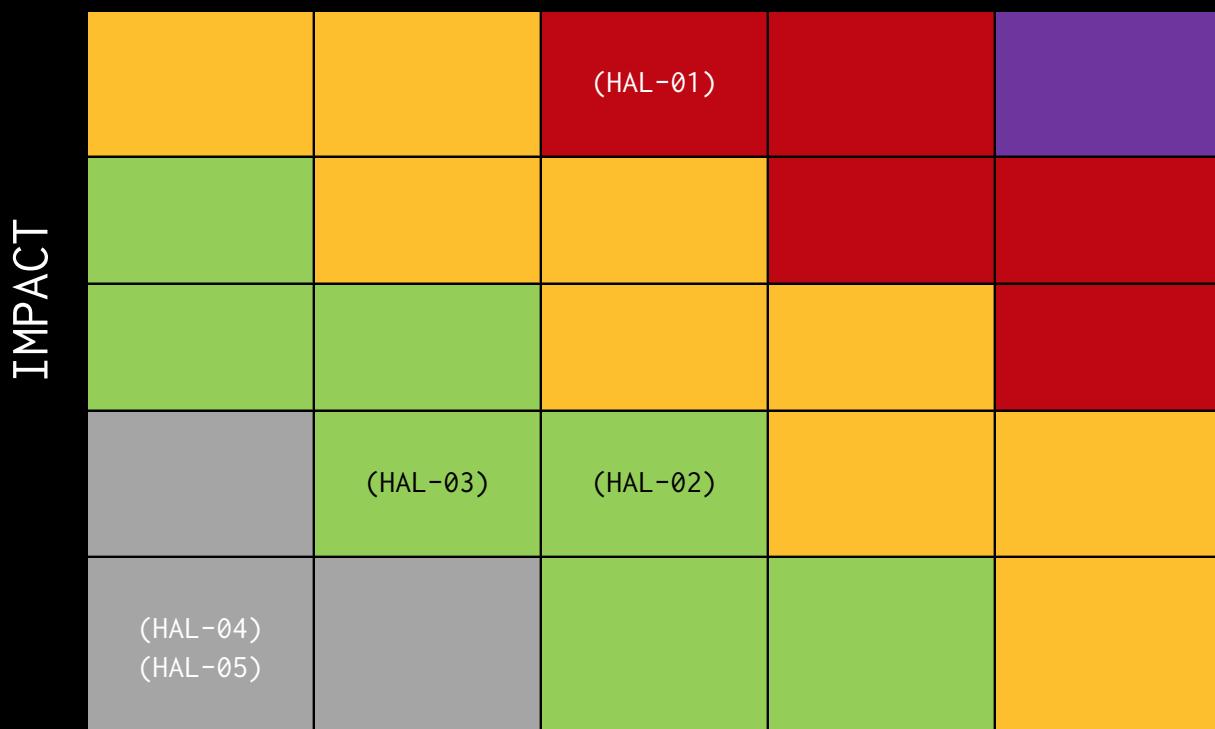
OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economical attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	2	2

LIKELIHOOD

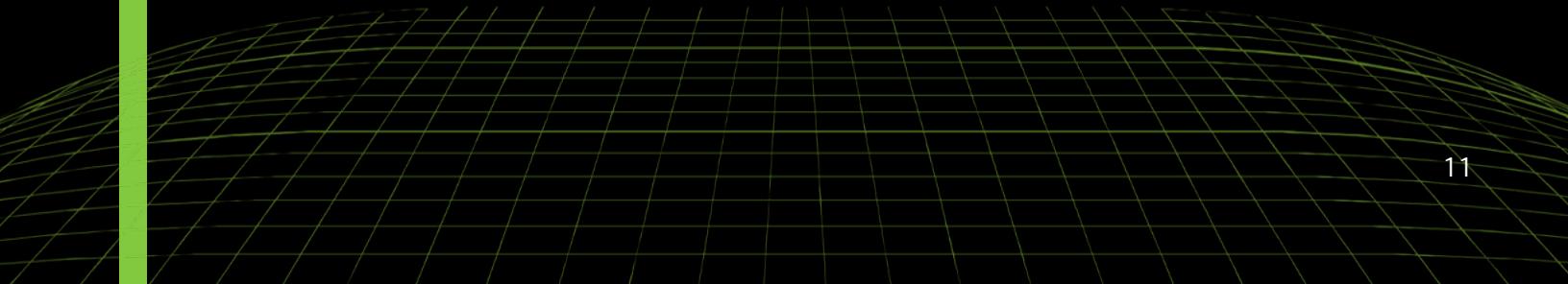


EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - PRODUCTTOKENHIGHBASE CONTRACT IS VULNERABLE TO SANDWICH ATTACKS	High	SOLVED - 01/05/2022
HAL-02 - MISSING ZERO ADDRESS CHECK	Low	SOLVED - 01/05/2022
HAL-03 - EXTERNAL CALLS WITHIN A LOOP	Low	SOLVED - 01/05/2022
HAL-04 - POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 01/05/2022
HAL-05 - TYPO IN FUNCTION NAME	Informational	SOLVED - 01/05/2022



FINDINGS & TECH DETAILS



3.1 (HAL-01) PRODUCTTOKENHIGHBASE CONTRACT IS VULNERABLE TO SANDWICH ATTACKS - HIGH

Description:

The contract `ProductTokenHighBase` is an ERC20 contract that allows users to buy and sell a product token. The buying and selling price of the token is based on a pricing algorithm called Bonding Curve. This algorithm causes the price of the token to increase with every purchase, meaning that early buyers are rewarded as they can, then, sell the tokens back for a profit.

The function `buy(uint256 maxPrice_)` allows users to buy a product token with a maximum acceptable price. It only allows buying 1 token per call:

Listing 1: ProductTokenHighBase.sol

```
78 function buy(uint256 maxPrice_) external virtual whenNotPaused
    nonReentrant {
79     if(endTime[FEATURE_ENDTIME_BUY] != 0) {
80         require(now256() < endTime[FEATURE_ENDTIME_BUY], "sale is
81             expire");
82     }
83     require(maxPrice_ > 0, "invalid max price");
84     transferFromHighToken(_msgSender(), address(this), maxPrice_);
85
86     (uint256 change) = _buy(maxPrice_);
87     if(change > 0) {
88         transferHighToken(_msgSender(), change);
89     }
90 }
```

The function `sell(uint32 amount_)` allows users to sell a product token, returning the equivalent HIGH tokens:

Listing 2: ProductTokenHighBase.sol

```

100 function sell(uint32 amount_) external virtual whenNotPaused
101     nonReentrant {
102     if(endTime[FEATURE_ENDTIME_SELL] != 0) {
103         require(now256() < endTime[FEATURE_ENDTIME_SELL], "sale is
104             expire");
105         require(amount_ > 0, "Amount must be non-zero.");
106         require(balanceOf(_msgSender()) >= amount_, "Insufficient tokens
107             .");
108         uint256 price = _sell(amount_);
109         transferHighToken(_msgSender(), price);
110     }

```

The growth of the price with each purchase is determined by the different bonding curve parameters, which are set in the `initialize()` function of the `ProductTokenHighBase` contract:

Listing 3: ProductTokenHighBase.sol (Lines 47-50)

```

27 /**
28 * @dev initializer function.
29 *
30 * @param _name the name of this token
31 * @param _symbol the symbol of this token
32 * @param _bondingCurve bonding curve instance address
33 * @param _productNft product nft instance address
34 * @param _reserveRatio the reserve ratio in the curve function.
35 *     Number in parts per million
36 * @param _maxTokenCount the amount of token that will exist for
37 *     this type.
38 * @param _supplyOffset this amount is used to determine initial
39 *     price.
40 * @param _baseReserve the base amount of reserve tokens, in
41 *     accordance to _supplyOffset.
42 * @param _endTime end time is the last time when user can buy and
43 *     sell;
44 *
45 */
46 function initialize(
47     string memory _name,

```

```
43     string memory _symbol,
44     address _high,
45     address _bondingCurve,
46     address _productNft,
47     uint32 _reserveRatio,
48     uint32 _maxTokenCount,
49     uint32 _supplyOffset,
50     uint256 _baseReserve,
51     uint256 _endTime
52 ) public virtual initializer{
53     HIGH = _high;
54     ProductTokenCore.initialize(
55         _name,
56         _symbol,
57         _bondingCurve,
58         _productNft,
59         _reserveRatio,
60         _maxTokenCount,
61         _supplyOffset,
62         _baseReserve
63     );
64     if(_endTime > 0) {
65         updateEndTime(FEATURE_ENDTIME_MAX, _endTime);
66     }
67 }
```

As there are no restriction or cooldown period to sell the product token after its purchase, the `buy()` function calls are vulnerable to a sandwich attack.

Proof of Concept:

Based in the parameters found in the `2_deploy_contract.js` file:

- `_reserveRatio` the reserve ratio in the curve function. Number in parts per million -> 30000
- `_maxTokenCount` the amount of token that will exist for this type. -> 250
- `_supplyOffset` this amount is used to determine initial price. -> 700
- `_baseReserve` the base amount of reserve tokens. -> 800

1. Attacker scans the mempool for 2 `ProductTokenHighBase.buy()` transactions.
2. Attacker detects 2 `ProductTokenHighBase.buy()` txs in the mempool coming from user1 and user2.
3. Attacker frontruns both txs and calls himself `ProductTokenHighBase.buy()`.
4. If both users had set the `maxPrice_` in `ProductTokenHighBase.buy()` to the exact price (`ProductTokenHighBase.getCurrentPrice()`), their txs would revert. If that was the case, it makes sense to think that they would resubmit their transactions with a higher `maxPrice_`. On the other hand, if both users had set the `maxPrice_` in `ProductTokenHighBase.buy()` with some margin, their txs would be completed but they would be paying a higher price for the `ProductTokenHighBase` product token.
5. Attacker backruns both txs (user1 and user2 txs) and sells the product token bought initially, getting a 3.33% profit:

Tokens paid initially by the attacker (during the frontrun):

`40_547860657636861605`

Tokens paid by user1: `42_461906982198102077`

Tokens paid by user2: `44_463383637847924478`

Total tokens received by the attacker after the sale of the product token (during the backrun): `41_898188427972082682`

Profitted tokens after sandwiching 2 users = `1_350327770335221077`

% of profit: $(1_350327770335221077 / 40_547860657636861605) \times 100 = 3.33\%$

With these bonding curve parameters, two additional purchases are needed before selling to make profit. With different bonding curve parameters, it would be possible to profit by just sandwiching one transaction instead of two, increasing this attack likelihood.

Risk Level:

Likelihood - 3

Impact - 5

Recommendation:

It is recommended to set the bonding curve parameters of the `ProductTokenHighBase` contract in order that, before a user can profit by selling the product token, a number high enough of buy orders are needed.

On the other hand, perhaps a simpler approach, another possible solution would be to add a cooldown on the `sell()` function, not allowing users to sell right away after a `buy()` call.

Remediation Plan:

SOLVED: The `HighStreetMarket` team solved the issue in the following commit IDs:

- `3d83211d1720c6cd1ae1cd21631a400c4932bad7`
- `27d266184835bdf54aa3c8135a7d2a2a89fc7b37`

A cooldown has been added that restricts users from calling `sell()`, `transferFrom()` and `transfer()` functions for a fixed period of time.

3.2 (HAL-02) MISSING ZERO ADDRESS CHECK - LOW

Description:

The contract `ProductTokenHighBase` is missing zero address checks in its `initialize()` function. Every address should be validated and checked that is different from zero.

Code location:

```
Listing 4: ProductTokenHighBase.sol

41 function initialize(
42     string memory _name,
43     string memory _symbol,
44     address _high,
45     address _bondingCurve,
46     address _productNft,
47     uint32 _reserveRatio,
48     uint32 _maxTokenCount,
49     uint32 _supplyOffset,
50     uint256 _baseReserve,
51     uint256 _endTime
52 ) public virtual initializer{
53     HIGH = _high;
54     ProductTokenCore.initialize(
55         _name,
56         _symbol,
57         _bondingCurve,
58         _productNft,
59         _reserveRatio,
60         _maxTokenCount,
61         _supplyOffset,
62         _baseReserve
63     );
64     if(_endTime > 0) {
65         updateEndTime(FEATURE_ENDTIME_MAX, _endTime);
66     }
67 }
```

Risk Level:

Likelihood - 3

Impact - 2

Recommendation:

It is recommended to validate that every address input is different from zero.

Remediation Plan:

SOLVED: The HighStreetMarket team solved the issue in the commit ID [84746701f0463a77d75c0ec8d7444784e594e42d](#)

3.3 (HAL-03) EXTERNAL CALLS WITHIN A LOOP - LOW

Description:

In the contract `ProductTokenHighBase` the function `tradein()` allows users to trade in their Product token for a NFT.

Listing 5: ProductTokenHighBase.sol (Lines 122)

```

116 function tradein(uint32 amount_) external virtual whenNotPaused
    nonReentrant{
117     if(endTime[FEATURE_ENDTIME_TRADEIN] != 0) {
118         require(now256() < endTime[FEATURE_ENDTIME_TRADEIN], "sale is
            expire");
119     }
120     require(amount_ > 0, "Amount must be non-zero.");
121     require(balanceOf(_msgSender()) >= amount_, "Insufficient tokens
            .");
122     _tradein(amount_);
123 }
```

Listing 6: ProductTokenCore.sol (Lines 466)

```

453 function _tradein(uint32 amount_) internal virtual {
454
455     (, uint256 fee) = _sellReturn(amount_);
456
457     _burn(_msgSender(), amount_);
458     // redeem value should give to supplier
459     uint256 tradinReturn = _tradinReturn(amount_);
460
461     // 50% of fee and redeem price are for supplier
462     _updateSupplierFee(fee * FEE_MULTIPLIER / FEE_RATE_IN_SELL +
        tradinReturn);
463     // 50% of fee, is for plateform
464     _updatePlateformFee(fee * (FEE_RATE_IN_SELL - FEE_MULTIPLIER) /
        FEE_RATE_IN_SELL);
465
466     nft.mintBatch(_msgSender(), tradeinCount, amount_);
467 }
```

```
468     tradeinCount = tradeinCount + amount_;
469     tradeinReserveBalance = tradeinReserveBalance + tradinReturn;
470
471     emit Tradein(_msgSender(), amount_, tradinReturn, fee);
472 }
```

If a user trades in 10 Product tokens, 10 NFTs will be minted through the function `ProductNft.mintBatch()`:

Listing 7: ProductNft.sol (Lines 145)

```
140    function mintBatch(address to_, uint256 start_, uint256 count_)
141        external {
142        require(minters[_msgSender()] == true, 'permission denied');
143        require(to_ != address(0), 'invalid receiver');
144        require(start_ + count_ < maxCount, "cap exceeded");
145        for(uint256 index = start_; index < start_ + count_; index++)
146        {
147            _safeMint(to_, index);
148        }
149    }
```

The gas cost of minting 10 NFTs is 1.258.049. This would already exceed, for example, Metamask default gas limit of 1,200,000 GWEI. Users not aware of this may submit the transaction to then find out that the transaction have failed with an `Out of gas` message. A transaction of 1,200,000 GWEI in gas is worth \$451.95 (date: 01/01/2022). This is the potential loss that can happen to a user not aware of this.

For that reason, it is recommended to limit the `amount_` parameter of the `tradein()` function.

Proof of Concept:

```

Calling -> contract_ProductTokenHighBase.tradein(10, {'from': user1})
Transaction sent: 0x0ef9e8a0c66c7424fc520a2f6e6228d4d83f7128992blf86198c43la736e0840
  Gas price: 0.0 gwei  Gas limit: 6721975 Nonce: 21
    ProductTokenHighBase.tradein confirmed  Block: 13921317  Gas used: 1258049 (18.72%)

Call trace for '0x0ef9e8a0c66c7424fc520a2f6e6228d4d83f7128992blf86198c43la736e0840':
Initial call cost [-12996 gas]
ProductTokenHighBase.tradein 0:43409 [30062 / 1181045 gas]
  ERC20Upgradeable.balanceOf 251:19551 [3214 / 78733 gas]
    ProductTokenCore._tradein 281:403 [38 / 2798 gas]
      ProductTokenCore._sellReturn 287:398 [960 / 2760 gas]
        ProductTokenCore._getTotalSupply 304:366 [1800 gas]

    BancorBondingCurve.calculateSaleReturn [STATICCALL] 479:19422 [1715 / 72721 gas]
      address: contract_BancorBondingCurve.address
      input arguments:
        _supply: 710
        _reserveBalance: 1283611532980591381821
        _reserveRatio: 30000
        sellAmount: 10
      return value: 483611532980591381825

      SafeMathUpgradeable.sub 709:736 [91 gas]
      Power.power 746:19271 [394 / 70624 gas]
        Power.ln 761:15535 [51424 gas]
          Power.findPositionInMaxExpArray 15585:16541 [9596 gas]
          Power.fixedExp 16582:19258 [9210 gas]
        SafeMathUpgradeable.mul 19283:19317 [116 gas]
        SafeMathUpgradeable.sub 19333:19360 [91 gas]
        SafeMathUpgradeable.div 19364:19388 [84 gas]
      ERC20Upgradeable._burn 19566:19695 [-16173 gas]
      ProductTokenCore._tradinReturn 19701:19764 [1003 gas]

    BancorBondingCurve.calculatePriceForNTokens [STATICCALL] 19882:38215 [1735 / 70742 gas]
      address: contract_BancorBondingCurve.address
      input arguments:
        _supply: 710
        _reserveBalance: 80000000000000000000000000000000
        _reserveRatio: 30000
        amount: 10
      return value: 475150887200091539313

      SafeMathUpgradeable.add 20107:20135 [94 gas]
      Power.power 20145:38112 [394 / 68797 gas]
        Power.ln 20160:34376 [49597 gas]
          Power.findPositionInMaxExpArray 34426:35382 [9596 gas]
          Power.fixedExp 35423:38099 [9210 gas]
        SafeMathUpgradeable.mul 38127:38161 [116 gas]
      ProductTokenCore._updateSupplierFee 38344:38372 [5890 gas]
      ProductTokenCore._updatePlateformFee 38440:38468 [5890 gas]

ProductNft.mintBatch [CALL] 38549:43249 [1004898 gas]
  address: contract_ProductNft.address
  value: 0
  input arguments:
    to_: user1.address
    start_: 0
    count_: 10

```

Risk Level:

Likelihood - 2

Impact - 2

Recommendation:

It is recommended to limit the `amount_` parameter of the `tradein()` function, making sure that is, for example, lower than 10.

Remediation Plan:

SOLVED: The `HighStreetMarket team` solved the issue in the commit ID `56f794ef686c2840afd35f7255ce8cb23bba8127`

3.4 (HAL-04) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the following contracts there are functions marked as `public` but they are never directly called within the same contract or in any of their descendants:

`BancorBondingCurve.sol`

- `calculatePriceForNTokens()` (`BancorBondingCurve.sol#38-64`)
- `calculatePurchaseReturn()` (`BancorBondingCurve.sol#81-109`)
- `calculateSaleReturn()` (`BancorBondingCurve.sol#125-159`)

`ProductTokenCore.sol`

- `getCurrentPrice()` (`ProductTokenCore.sol#260-263`)
- `calculateBuyReturn()` (`ProductTokenCore.sol#329-334`)
- `calculateSellReturn()` (`ProductTokenCore.sol#363-367`)

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

If the functions are not intended to be called internally or by their descendants, it is better to mark all of these functions as `external` to reduce gas costs.

Remediation Plan:

SOLVED: The `HighStreetMarket` team solved the issue in the commit ID `674c5f7a814acf33bafaedd43c94df0731e69c73`

3.5 (HAL-05) TYPO IN FUNCTION NAMES AND EVENTS - INFORMATIONAL

Description:

The following functions, variables, and events in the `ProductTokenCore` contract have a typo: `Plateform` instead of `Platform`.

- Line 64: `uint256 private plateformFee;`
- Line 150: `event ClaimPlateformFee(address indexed sender, uint256 amount);`
- Line 411:
`_updatePlateformFee(fee * (FEE_RATE_IN_BUY - FEE_MULTIPLIER)/
FEE_RATE_IN_BUY);`
- Line 440:
`_updatePlateformFee(fee * (FEE_RATE_IN_SELL - FEE_MULTIPLIER)/
FEE_RATE_IN_SELL);`
- Line 464:
`_updatePlateformFee(fee * (FEE_RATE_IN_SELL - FEE_MULTIPLIER)/
FEE_RATE_IN_SELL);`
- Line 488:
`function _updatePlateformFee(uint256 fee_)internal virtual {`
- Line 489: `plateformFee = plateformFee + fee_;`
- Line 551: `function getPlateformFee()external view virtual returns(
uint256 amount){`
- Line 553: `return plateformFee;`
- Line 562:
`function claimPlateformFee(uint256 amount_)external virtual {`
- Line 564: `require(amount_ <= plateformFee, "amount is exceed");`
- Line 567: `plateformFee = plateformFee - amount_;`
- Line 569: `emit ClaimPlateformFee(_msgSender(), amount_);`

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to correct the typo in all the functions, variables, and events mentioned.

Remediation Plan:

SOLVED: The `HighStreetMarket team` solved the issue in the commit ID `97746779dd28d751e1d1ce87b2c123300928704c`

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped 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.

Slither results:

BancorBondingCurve.sol

Power.sol

ProductNft.sol

```

Variable "ERC721,_checkOnERC721Received(address,address,uint256,bytes)" (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Variable "ERC721,_checkOnERC721Received(address,address,uint256,bytes)" (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374) in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374) uses assembly
Variable "ERC721,_checkOnERC721Received(address,address,uint256,bytes).reason" (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379) in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379) potentially used before declaration reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379)
Variable "ERC721,_checkOnERC721Received(address,address,uint256,bytes)" (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379) in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379) potentially used before declaration revert(uint256,uint256)(0x1 + reason.length) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#383)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-use-of-local-variables

ERC721,_checkOnERC721Received(address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#382-384)
Address.isContract(addresses) (node_modules/@openzeppelin/contracts/utils/Address.sol#22-24) uses assembly
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#195-215) uses assembly
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ProductNft.mint(address,uint256) (contract/ProductNft.sol#124-129) compares to a boolean constant;
- require(bool,string)minters[_msgSender()] == true,permission denied (contract/ProductNft.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#bool-equality

Different versions of Solidity is used:
- Version used: "[>0.8.0,<=0.8.0]"
- > 0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#19)
- > 0.8.0 (node_modules/@openzeppelin/contracts/contract/tokens/ERC721/ERC721.sol#18)
- > 0.8.0 (node_modules/@openzeppelin/contracts/contract/tokens/ERC721/ERC721.sol#19)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721Receiver.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/extension/ERC721Enumerable.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/extension/ERC721Enumerable.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/extension/ERC721Metadata.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/Address.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/Context.sol#1)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/introspection/IERC165.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/introspection/IERC165.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/introspection/IERC165.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragme-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#91) is never used and should be removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#95) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#105-114) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#118-127) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#118-127) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#148-160) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#148-160) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-190) is never used and should be removed
Address.sendValue(address,uint256,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-190) is never used and should be removed
Context._msgData (node_modules/@openzeppelin/contracts/context/Context.sol#22) is never used and should be removed
ERC721._baseURI (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721.sol#104-106) is never used and should be removed
ERC721._burn(uint256) (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721.sol#304-316) is never used and should be removed
String.collectString(uint256) (node_modules/@openzeppelin/contract/utils/Strings.sol#19-25) is never used and should be removed
String.collectString(uint256,uint256) (node_modules/@openzeppelin/contract/utils/Strings.sol#55-65) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#read-code

Pragma version<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#103) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version<0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721.sol#103) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
ERC721,_checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Variable "ERC721,_checkOnERC721Received(address,address,uint256,bytes)" (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374) in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374) potentially used before declaration reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379)
Variable "ERC721,_checkOnERC721Received(address,address,uint256,bytes).reason" (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379) in ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#379) potentially used before declaration revert(uint256,uint256)(0x1 + reason.length) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#383)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-use-of-local-variables

ERC721,_checkOnERC721Received(address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#382-384)
Address.isContract(addresses) (node_modules/@openzeppelin/contracts/utils/Address.sol#22-24) uses assembly
Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#195-215) uses assembly
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ProductNft.mint(address,uint256) (contract/ProductNft.sol#124-129) compares to a boolean constant;
- require(bool,string)minters[_msgSender()] == true,permission denied (contract/ProductNft.sol#141)
- require(bool,string)minters[_msgSender()] == true,permission denied (contract/ProductNft.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#bool-equality

Different versions of Solidity is used:
- Version used: "[>0.8.0,<=0.8.0]"
- > 0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#19)
- > 0.8.0 (node_modules/@openzeppelin/contracts/contract/tokens/ERC721/ERC721.sol#18)
- > 0.8.0 (node_modules/@openzeppelin/contracts/contract/tokens/ERC721/ERC721.sol#19)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721Receiver.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/extension/ERC721Enumerable.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/extension/ERC721Enumerable.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/extension/ERC721Metadata.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/Address.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/Context.sol#1)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/introspection/IERC165.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/introspection/IERC165.sol#3)
- > 0.8.0 (node_modules/@openzeppelin/contract/utils/introspection/IERC165.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragme-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#91) is never used and should be removed
Address.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#95) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#105-114) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#118-127) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#118-127) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#148-160) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#148-160) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-190) is never used and should be removed
Address.sendValue(address,uint256,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-190) is never used and should be removed
Context._msgData (node_modules/@openzeppelin/contracts/context/Context.sol#22) is never used and should be removed
ERC721._baseURI (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721.sol#104-106) is never used and should be removed
ERC721._burn(uint256) (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721.sol#304-316) is never used and should be removed
String.collectString(uint256) (node_modules/@openzeppelin/contract/utils/Strings.sol#19-25) is never used and should be removed
String.collectString(uint256,uint256) (node_modules/@openzeppelin/contract/utils/Strings.sol#55-65) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#read-code

Pragma version<0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version<0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#103) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version<0.8.0 (node_modules/@openzeppelin/contract/tokens/ERC721/ERC721.sol#103) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

```

ProductTokenCore.sol

```

ContractUpgradable._upgradeable (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#18) shadows:
- ContractUpgradable._upgradeable (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/ContextUpgradeable.sol#18)
ContractUpgradable._upgradeable_gmp (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/security/PausableUpgradeable.sol#19) shadows:
- ContractUpgradable._upgradeable_gmp (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/ContextUpgradeable.sol#19)
ERC20Upgradeable._upgradeable (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/math/SafeMathUpgradeable.sol#19) shadows:
- ContractUpgradable._upgradeable (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/ContextUpgradeable.sol#19)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

Power._inuint256(uint256) (contracts/Power.sol#205-220) performs a multiplication on the result of a division:
- x = _numerical / FIXED_1 // denominator (contracts/Power.sol#209)

ProductTokenCore._buyToken(uint256) (contract/ProductTokenCore.sol#307-321) performs a multiplication on the result of a division:
- value = amountReserve * FEE_DIVIDER / (FEE_DIVIDER + FEE RATE IN BUY) (contract/ProductTokenCore.sol#310)
- feeRate = FEE RATE IN BUY / FEE DIVIDER (contract/ProductTokenCore.sol#312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

Reentrancy in ProductTokenCore._tradeInReturn(uint256) (contract/ProductTokenCore.sol#450-473):
- External calls:
  - nft.mintBatch(_msgSender(), tradeInReturn, amount) (contract/ProductTokenCore.sol#466)
- State variable:
  - tradeInReturn (contract/ProductTokenCore.sol#468)
  - tradeInReturn = tradeInReturn + tradeInReturn (contract/ProductTokenCore.sol#469)
- tradeInReturnBalance = tradeInReturnBalance + tradeInReturn (contract/ProductTokenCore.sol#469)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ERC721,_checkOnERC721Received(addresses,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#369-390) ignores return value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#374)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

ProductTokenCore.initialize(string,string,address,uint32,uint32,uint256) (contract/ProductTokenCore.sol#175) shadows:
- ERC20Upgradeable._symbol (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/math/SafeMathUpgradeable.sol#19) shadows:
  - ERC20Upgradeable._symbol (node_modules/@openzeppelin/contracts-upgradeable/contracts-upgradeable/math/SafeMathUpgradeable.sol#19) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

```

AUTOMATED TESTING

AUTOMATED TESTING

ProductTokenHighBase.sol

AUTOMATED TESTING

- No major issues were found by Slither. All the reentrancies flagged by Slither are false positives. The functions `ProductTokenHighBase.buy()`, `ProductTokenHighBase.sell()` and `ProductTokenHighBase.tradein()` are all protected by the `nonReentrant` modifier.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

BancorBondingCurve.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Power.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

ProductNft.sol

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

ProductTokenCore.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

ProductTokenHighBase.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

- No major issues were found by MythX. The floating pragma is a false positive as the pragma is set in the `truffle-config.js` file to the

AUTOMATED TESTING

0.8.10 version.

THANK YOU FOR CHOOSING
 HALBORN