WOLFYStreetBets

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 12th-19th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/12/2021	Gokberk Gulgun
0.9	Document Edits	08/17/2021	Gokberk Gulgun
1.0	Final Version	08/18/2021	Gabi Urrutia
1.1	Remediation Plan	08/19/2021	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

WOLFY engaged Halborn to conduct a security assessment on their Smart contract beginning on August 12th, 2021 and ending August 19th, 2021.

The security assessment was scoped to the smart contract WolfyStreetBetsv1.sol. Halborn conducted this audit to measure security risk and identify any vulnerabilities introduced during the final stages of development before the WOLFYStreetBets production release.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverable set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided 1 week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is 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 to achieve the following:

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

In summary, Halborn identified few security risks addressed by WOLFY 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 the smart contract audit.While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts 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 read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of use and safety of the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Testnet deployment (Truffle, Ganache)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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.
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:

https://mumbai.polygonscan.com/address/0x9112a755E119E5CEd42FB496f679E55B996adE67#code

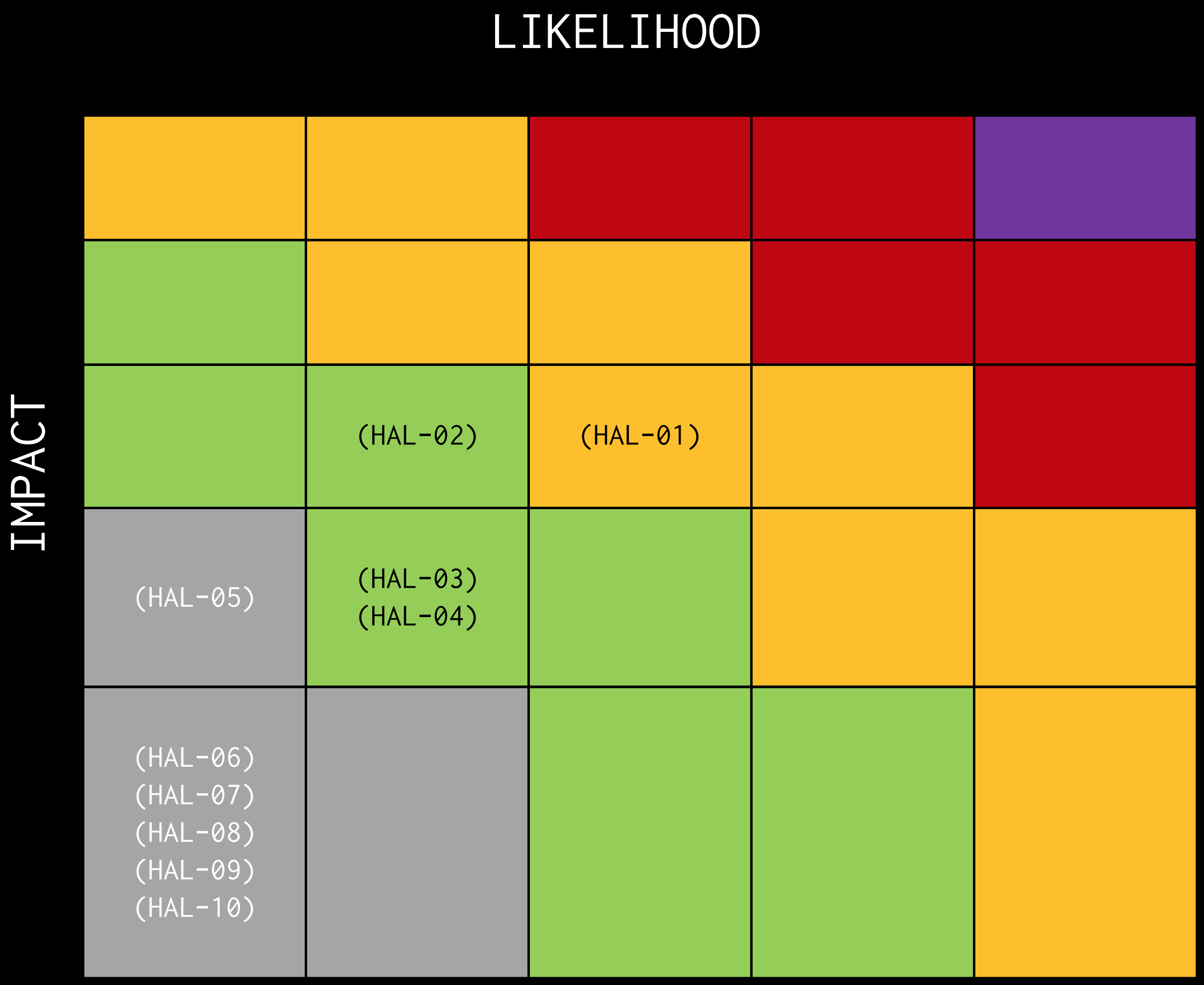
FIXED COMMIT ID: e04c22d0c5467ba60a50b9fd21e3d8f902719266

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economics attacks.

However, if any economic issue is found, it will be marked as an INFORMATIONAL. This report identified several items that are economic in nature, (such as the way Liquidity can be accessed by owners) but may not be considered vulnerabilities in the context for this scope.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	6



🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨	🟥	🟥
🟩	🟩 (HAL-02)	🟨 (HAL-01)	🟨	🟥
⬜ (HAL-05)	🟩 (HAL-03) (HAL-04)	🟩	🟨	🟨
⬜ (HAL-06) (HAL-07) (HAL-08) (HAL-09) (HAL-10)	⬜	🟩	🟩	🟨

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INTEGER OVERFLOW	Medium	SOLVED - 08/18/2021
DIVIDE BEFORE MULTIPLY	Low	SOLVED - 08/18/2021
LACK OF FUNCTIONALITY ON THE TRUSTED FORWARDER FUNCTION	Low	SOLVED - 08/18/2021
GAS IMPROVEMENT ON THE WITHDRAW LIQUIDITY FUNCTION	Low	SOLVED - 08/18/2021
MISSING EVENTS EMITTING	Low	SOLVED - 08/18/2021
IMPROPER CHECK EFFECT INTERACTION PATTERN USAGE	Informational	SOLVED - 08/18/2021
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	SOLVED - 08/18/2021
IMPROPER INPUT VALIDATION ON THE PREDICTION ASSETS	Informational	SOLVED - 08/18/2021
REDUNDANT STATEMENT ON THE REWARD MANAGER	Informational	ACKNOWLEDGED
MISUSE OF GAS ON THE PAYOUTOWNERLIQUIDITY FUNCTION	Informational	SOLVED - 08/18/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) INTEGER OVERFLOW - MEDIUM

Description:

An overflow happens when an arithmetic operation reaches the maximum size of a type. In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits -- either larger than the maximum or lower than the minimum re-presentable value.

Code Location:

WolfyStreetBetsV1.sol - Line #670-671

Listing 1: WolfyStreetBetsV1.sol

```
function startPool(uint256 _predictionAsset1, uint256 _predictionAsset2) public onlyOwner {
        require(!poolStarted, "Previous pool not finalized yet");
        require(totalLiquidityLowRisk + liquidityDetailsOwner._lowRisk > 0 ," Low Risk Pool: Please add liquidity");
        require(totalLiquidityHighRisk + liquidityDetailsOwner._highRisk > 0 ," High Risk Pool: Please add liquidity");
        // 30 minutes for testing ONLY
        require(block.timestamp > liquidityCycleStartTime.add(30 minutes), "Cannot start pool during liquidity cycle");
        // PLEASE UNCOMMENT ME FOR MAINNET
        // require(block.timestamp > liquidityCycleStartTime.add(12 hours), "Cannot start pool during liquidity cycle");

        predictionAsset1 = _predictionAsset1;
        predictionAsset2 = _predictionAsset2;

        poolStartTime = block.timestamp;
        poolStarted = true;

        liquidityCycle = false;
```

```
    }
```

WolfyStreetBetsV1.sol - Line #728-729-730-733-734-735-736-739

Listing 2: WolfyStreetBetsV1.sol

```
function stake(uint256 _amount, bool _isLowRisk) public {
        require(_amount > 0 , "You can't stake with 0. Choose an amount!");
        require(poolStarted, "Cannot stake until pool has been started!");
        // FOR MAINNET PLEASE UNCOMMENT ME
        // require(block.timestamp <= poolStartTime.add(12 hours),"12 hour staking window has now passed!" ); // Can stake upto 12 hours from start pool.

        uint256 stakeAmount;

        if (liquidityReward >= 1) {
          stakeAmount = _amount.sub(((_amount.mul(liquidityReward)).div(100)).div(10));
        }
        else {
          stakeAmount = _amount;
        }
        if (_isLowRisk) {
            require(ledgerL.add(stakeAmount) <= (totalLiquidityLowRisk + liquidityDetailsOwner._lowRisk).mul(6), "Low risk pool: Staking limit reached!");
            require(_poolBalances[_msgSender()][_isLowRisk].add(stakeAmount) <= (totalLiquidityLowRisk + liquidityDetailsOwner._lowRisk).mul(6), "Low risk pool: Staking limit reached!");
            liquidityRewardCollectedLowRisk += ((_amount.mul(liquidityReward)).div(100)).div(10);
            ledgerL += stakeAmount;
        }
        else {
            require(ledgerH.add(stakeAmount) <= (totalLiquidityHighRisk + liquidityDetailsOwner.
                    _highRisk).mul(3), "High risk pool: Staking limit
                    reached!");
                require(_poolBalances[_msgSender()][_isLowRisk].add(
                    stakeAmount) <= (totalLiquidityHighRisk +
                    liquidityDetailsOwner._highRisk).mul(3), "High risk
                     pool: Staking limit reached!");
                liquidityRewardCollectedHighRisk += ((_amount.mul(
                    liquidityReward)).div(100)).div(10);
                ledgerH += stakeAmount;
            }
            _isLowRisk == true ? storeUsers(_msgSender(),
               _lowRiskUsers): storeUsers(_msgSender(), _highRiskUsers
               );
            _poolBalances[_msgSender()][_isLowRisk] += stakeAmount;
            TOKEN.safeTransferFrom(_msgSender(), address(this),
               _amount);

            distributeLiquidityRewards(true);
            distributeLiquidityRewards(false);
        }

```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider to replace all + - * / mathematical operations via Safe Math library implementations. (**add-sub-mul-div**) It is recommended to use vetted safe math libraries for arithmetic operations consistently throughout the smart contract system or use pragma version bigger than 0.8.0 that adds arithmetic checks automatically.

Remediation Plan:

SOLVED: WOLFY Team rightly implemented mathematical operations. All mathematical operations are completed through SafeMath.

3.2 (HAL-02) DIVIDE BEFORE MULTIPLY - LOW

Description:

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision. In this audit, there are multiple instances found where division is being performed before multiplication operation in the WolfyStreetBetsV1.sol.

Code Location:

WolfyStreetBetsV1.sol Line #1019-1025

Listing 3: EglContract.sol (Lines)

```
1016 function  distributeLiquidityRewards(bool _isLowRisk) internal {
1017         for (uint i=0;i<LiquidityDetailsRecord.length;i++) {
1018             if (_isLowRisk && LiquidityDetailsRecord[i].isLowRisk
                     == true) {
1019                 uint256 rewardPerc = ((LiquidityDetailsRecord[i].
                         _lowRisk.mul(10**decimals())).mul(100*10**
                         decimals())).div(totalLiquidityLowRisk.mul(10**
                         decimals()));
1020                 uint256 rewardAmount = ((
                         liquidityRewardCollectedLowRisk.mul(10**
                         decimals()) ).mul(rewardPerc) )/(100 * 10**
                         decimals());
1021                 rewardPaidRecord[LiquidityDetailsRecord[i].
                         _address].push(RewardPaid(rewardAmount.div(10**
                         decimals()),block.timestamp,false));
1022                 userLPReward[LiquidityDetailsRecord[i]._address]
                         += rewardAmount;
1023             }
1024             else if (_isLowRisk == false && LiquidityDetailsRecord
                     [i].isLowRisk == false) {
1025                 uint256 rewardPerc = ((LiquidityDetailsRecord[i].
                         _highRisk.mul(10**decimals())).mul(100*10**
                         decimals())).div(totalLiquidityHighRisk.mul
                         (10**decimals()));
```

```
                    uint256 rewardAmount = ((liquidityRewardCollectedHighRisk.mul(10**decimals()) ).mul(rewardPerc) )/(100 * 10**decimals());
                    rewardPaidRecord[LiquidityDetailsRecord[i]._address].push(RewardPaid(rewardAmount.div(10**decimals()),block.timestamp,false));
                    userLPReward[LiquidityDetailsRecord[i]._address] += rewardAmount;
                }
            }
            if (_isLowRisk) {
                liquidityRewardCollectedLowRisk = 0;
            }
            else {
                liquidityRewardCollectedHighRisk = 0;
            }

        }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Consider doing multiplication operation before division to prevail precision in the values in non floating data type.

Remediation Plan:

SOLVED: WOLFY Team rightly implemented mathematical operations. The calculation is adjusted according to suggestion.

3.3 (HAL-03) LACK OF FUNCTIONALITY ON THE TRUSTED FORWARDER FUNCTION - LOW

Description:

In the WolfyStreetBetsV1.sol contract, trusted forwarder has been used for biconomy upgrades. However, the function didn't set trusted forwarder there that is uncomplete.

Code Location:

Listing 4: WolfyStreetBetsV1.sol (Lines)

```
/**
* @dev Updates trusted forwarder should biconomy upgrades occur.
*/
function setTrustedForwarder(address _trustedForwarder) public view onlyOwner {
    require (_trustedForwarder != address(0), "Address cannot be 0x0");
    require (_trustedForwarder != address(this), "Address cannot be contract address");
}
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to set trusted forwarder on the related function. If that functionality will not use, for gas improvement the function should be deleted from the code base.

FINDINGS & TECH DETAILS

Listing 5: WolfyStreetBetsV1.sol (Lines)

```
/**
* @dev Updates trusted forwarder should biconomy upgrades occur.
*/
function setTrustedForwarder(address _trustedForwarder) public view onlyOwner {
    require (_trustedForwarder != address(0), "Address cannot be 0x0");
    require (_trustedForwarder != address(this), "Address cannot be contract address");
    trustedForwarder = _trustedForwarder;
    emit TrustedForwarderSet(trustedForwarder);
}

```

Remediation Plan:

SOLVED: WOLFY Team modified the code to set trusted forwarder on the related function.

3.4 (HAL-04) GAS IMPROVEMENT ON THE WITHDRAW LIQUIDITY FUNCTION - LOW

Description:

In the WolfyStreetBetsV1.sol contract, _isLowRisk variable is used for checking multiple conditions on the provideLiquidity function. However, that function redundantly checked multiple **if/else** statements. This implementation will spend more gas with multiple inner statemements.

Code Location:

WolfyStreetBetsV1.sol

Listing 6: WolfyStreetBetsV1.sol

```
    function provideLiquidity(uint256 amount, bool _isLowRisk)
        public {
        if (_msgSender() != owner()) {
            if (_isLowRisk) {
                LiquidityDetailsRecord.push(LiquidityDetails(
                    amount,0,_msgSender(),block.timestamp,true,
                    false,false));
            }
            else {
                LiquidityDetailsRecord.push(LiquidityDetails(0,
                    amount,_msgSender(),block.timestamp,false,false
                    ,false));
            }
            if (_isLowRisk == true) {
                totalLiquidityLowRisk += amount;
                currentLowLiquidity[_msgSender()] += amount;
                storeUsers(_msgSender(),LiquidityLRUsers);

            }
            else {
                totalLiquidityHighRisk += amount;
                currentHighLiquidity[_msgSender()] += amount;
                storeUsers(_msgSender(),LiquidityHRUsers);
            }

        }
```

Recommendation:

Consider to eliminate multiple condition check which proposes same inner statements.

Listing 7: WolfyStreetBetsV1.sol

```
function provideLiquidity(uint256 amount, bool _isLowRisk) public {
    if (_msgSender() != owner()) {
        if (_isLowRisk) {
            LiquidityDetailsRecord.push(LiquidityDetails(amount,0,_msgSender(),block.timestamp,true,false,false));
            totalLiquidityLowRisk += amount;
            currentLowLiquidity[_msgSender()] += amount;
            storeUsers(_msgSender(),LiquidityLRUsers);
        }
        else {
            LiquidityDetailsRecord.push(LiquidityDetails(0,amount,_msgSender(),block.timestamp,false,false,false));
            totalLiquidityHighRisk += amount;
            currentHighLiquidity[_msgSender()] += amount;
            storeUsers(_msgSender(),LiquidityHRUsers);
        }
    ....

    }
```

Remediation Plan:

SOLVED: WOLFY Team removed the excess of if/else statements.

3.5 (HAL-05) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

It has been observed that critical functionality is missing emitting event for some functions on the WolfyStreetBetsV1.sol contract. These functions should emit events after completing the transactions.

Code Location:

Listing 8: Missing Events

```
function setTrustedForwarder(address _trustedForwarder)
function setWinFactorL(uint256 _winFactorL)
function setWinFactorH(uint256 _winFactorH)
function storeUsers(address receiver, address[] storage arrayData)
function startPool(uint256 _predictionAsset1, uint256
    _predictionAsset2)
function stopPool(uint256 _predictionAsset1, uint256
    _predictionAsset2)
function stake(uint256 _amount, bool _isLowRisk)
function rewardManager(uint256 _factor, bool _res)
function withdrawPredictionStake(uint256 _amount, bool _isLowRisk)
function provideLiquidity(uint256 amount, bool _isLowRisk)
function withdrawLiquidityRewards(uint256 amount)
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendations:

Consider emitting an event when calling related functions on the list above.

Remediation Plan:

SOLVED: WOLFY Team added events on the functions.

3.6 (HAL-06) IMPROPER CHECK EFFECT INTERACTION PATTERN USAGE - INFORMATIONAL

Description:

In the Smart Contracts, The check effect interaction pattern is used to reduce the attack surface for malicious contracts trying to hijack control flow after an external call. In the WolfyStreetBetsV1.sol, ledgerL and ledgerH is updated after an external call.

Code Location:

Listing 9: WolfyStreetBetsV1.sol (Lines 118,119)

```
    function withdrawPredictionStake(uint256 _amount, bool _isLowRisk) public nonReentrant {
        require(!poolStarted, "Cannot withdraw Asset while pool is running");
        require(_msgSender() != owner());
        require(_amount <= _poolBalances[_msgSender()][_isLowRisk], "Insufficient Balance");
        _poolBalances[_msgSender()][_isLowRisk] -= _amount;
        TOKEN.safeTransfer(_msgSender(), _amount);

        if (_isLowRisk) {
            ledgerL -= _amount;
        }
        else {
            ledgerH -= _amount;
        }
    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

In the withdrawPredictionStake function, ledgerL and ledgerH should be updated before an external call.

Remediation Plan:

SOLVED: External call is currently done after ledgerL and ledgerH are updated.

3.7 (HAL-07) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In the public functions, array arguments are immediately copied to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, the function expects its arguments being located in memory when the compiler generates the code for an internal function.
Also, methods do not necessarily have to be public if they are only called within the contract-in such case they should be marked internal.

Affected Smart Contract Functions:

WolfyStreetBetsV1:
provideLiquidity,withdrawPredictionStake,stake,stopPool,startPool,setWinFactorL,setWinFactorH,setTrustedForwarder

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practice, you should use external if you expect that the function will only be called externally and use public if you need to call the function internally. To sum up, all can access to public functions, external functions only can be accessed externally and internal functions can only be called within the contract.

Remediation Plan:

SOLVED: WOLFY Team declared an external instead of public in the suggested functions.

3.8 (HAL-08) IMPROPER INPUT VALIDATION ON THE PREDICTION ASSETS - INFORMATIONAL

Description:

In the WolfyStreetBetsV1.sol contract, after providing liquidity to the pool, an owner can start the pool. The startPool function takes two argument named as _predictionAsset1 and _predictionAsset2. However, these function arguments are not validated.

Code Location:

Listing 10: WolfyStreetBetsV1.sol (Lines)

```
function startPool(uint256 _predictionAsset1, uint256 _predictionAsset2) public {
    require(!poolStarted, "Previous pool not finalized yet");
    require(totalLiquidityLowRisk + liquidityDetailsOwner._lowRisk > 0 ," Low Risk Pool: Please add liquidity");
    require(totalLiquidityHighRisk + liquidityDetailsOwner._highRisk > 0 ," High Risk Pool: Please add liquidity");
    // 30 minutes for testing ONLY
    require(block.timestamp > liquidityCycleStartTime.add(30 minutes), "Cannot start pool during liquidity cycle");
    // PLEASE UNCOMMENT ME FOR MAINNET
    // require(block.timestamp > liquidityCycleStartTime.add(12 hours), "Cannot start pool during liquidity cycle");

    predictionAsset1 = _predictionAsset1;
    predictionAsset2 = _predictionAsset2;

    poolStartTime = block.timestamp;
    poolStarted = true;

    liquidityCycle = false;
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to validate function arguments. To preventing miscalculation, these function arguments should be more than zero.

Remediation Plan:

SOLVED: WOLFY Team added validation in the function arguments.

3.9 (HAL-09) REDUNDANT STATEMENT ON THE REWARD MANAGER - INFORMATIONAL

Description:

In the WolfyStreetBetsV1.sol contract, the reward manager function adjusts pool balances due to risk score. On the function, previousResultRecord variable has been used for keeping record results. But, these records are not used in the contract. The redundant statements will spend gas more.

Code Location:

Listing 11: WolfyStreetBetsV1.sol (Lines)

```
    ... if (_factor >= winFactorL && _res == true) {
        // LR | PROFIT => +12%
        for (uint i = 0 ; i < _lowRiskUsers.length ; i++) {
          if (liquidityDetailsOwner._lowRisk >=
            _poolBalances[_lowRiskUsers[i]][true].mul(120).
            div(1000)) {
            liquidityDetailsOwner._lowRisk -=
              _poolBalances[_lowRiskUsers[i]][true].mul
              (120).div(1000);
            _profitStakers[_lowRiskUsers[i]][true] +=
              _poolBalances[_lowRiskUsers[i]][true].mul
              (120).div(1000);
            _poolBalances[_lowRiskUsers[i]][true] +=
              _poolBalances[_lowRiskUsers[i]][true].mul
              (120).div(1000);
            previousResultRecord.push(ResultRecord(true,
              true));
          }
          else {
            liquidityDetailsOwner._lowRisk = 0;
            _profitStakers[_lowRiskUsers[i]][true] +=
              _poolBalances[_lowRiskUsers[i]][true].mul
              (120).div(1000);
            _poolBalances[_lowRiskUsers[i]][true] +=
              _poolBalances[_lowRiskUsers[i]][true].mul
```

```
                                (120).div(1000);
                            previousResultRecord.push(ResultRecord(true,
                                true));
                        }
                    }
        ...     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider to delete redundant statements from the contract.

Remediation Plan:

ACKNOWLEDGED: WOLFY Team claims that the use of previousResultRecord is intended in the code because it is returned in a view function for front-end proposes.

3.10 (HAL-10) MISUSE OF GAS ON THE PAYOUTOWNERLIQUIDITY FUNCTION - INFORMATIONAL

Description:

In the WolfyStreetBetsV1.sol contract, tempRecordForloss struct has been used temporarily for keeping LiquidityDetails on the function. However, this struct is redundant that cause unnecessary for loop.

Code Location:

WolfyStreetBetsV1.sol Line# 777-782

Listing 12: WolfyStreetBetsV1.sol (Lines)

```
777     ...   if (totalWinHighRisk >= liquidityDetailsOwner._highRisk)
            {
778             uint256 cloneTotalLiquidityHighRisk =
                    totalLiquidityHighRisk;
779             uint256 diffrenceTobePaidHighRisk = totalWinHighRisk.
                    sub(liquidityDetailsOwner._highRisk);
780             for (uint256 i=0;i<LiquidityHRUsers.length;i++) {
781                 uint256 deductionPercentage = (
                        currentHighLiquidity[LiquidityHRUsers[i]].mul
                        (100))/cloneTotalLiquidityHighRisk;
782                 uint256 amount = (diffrenceTobePaidHighRisk.mul(
                        deductionPercentage)).div(100);
783                 currentHighLiquidity[LiquidityHRUsers[i]] -=
                        amount;
784                 tempRecordForloss.push(LiquidityDetails(0,amount,
                        LiquidityHRUsers[i],block.timestamp,false,false
                        ,true));
785                 totalLiquidityHighRisk -= amount;
786             }
787             for (uint256 i=0;i<tempRecordForloss.length;i++) {
788                 LiquidityDetailsRecord.push(tempRecordForloss[i]);
789             }
790
```

```
        liquidityDetailsOwner._highRisk = 0;
        delete tempRecordForloss;

    }
....
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

The tempRecordForLost struct can be deleted from the contract. Instead of tempRecordForloss struct, record can be pushed into LiquidityDetailsRecord struct.

Listing 13: WolfyStreetBetsV1.sol (Lines)

```
...   if (totalWinHighRisk >= liquidityDetailsOwner._highRisk)
    {
        uint256 cloneTotalLiquidityHighRisk =
            totalLiquidityHighRisk;
        uint256 diffrenceTobePaidHighRisk = totalWinHighRisk.
            sub(liquidityDetailsOwner._highRisk);
        for (uint256 i=0;i<LiquidityHRUsers.length;i++) {
            uint256 deductionPercentage = (
                currentHighLiquidity[LiquidityHRUsers[i]].mul
                (100))/cloneTotalLiquidityHighRisk;
            uint256 amount = (diffrenceTobePaidHighRisk.mul(
                deductionPercentage)).div(100);
            currentHighLiquidity[LiquidityHRUsers[i]] -=
                amount;
            LiquidityDetailsRecord.push(LiquidityDetails(0,
                amount,LiquidityHRUsers[i],block.timestamp,
                false,false,true));
            totalLiquidityHighRisk -= amount;
        }
        liquidityDetailsOwner._highRisk = 0;
    }
    ....
```

Remediation Plan:

SOLVED: WOLFY Team rightly applied the suggested changes.

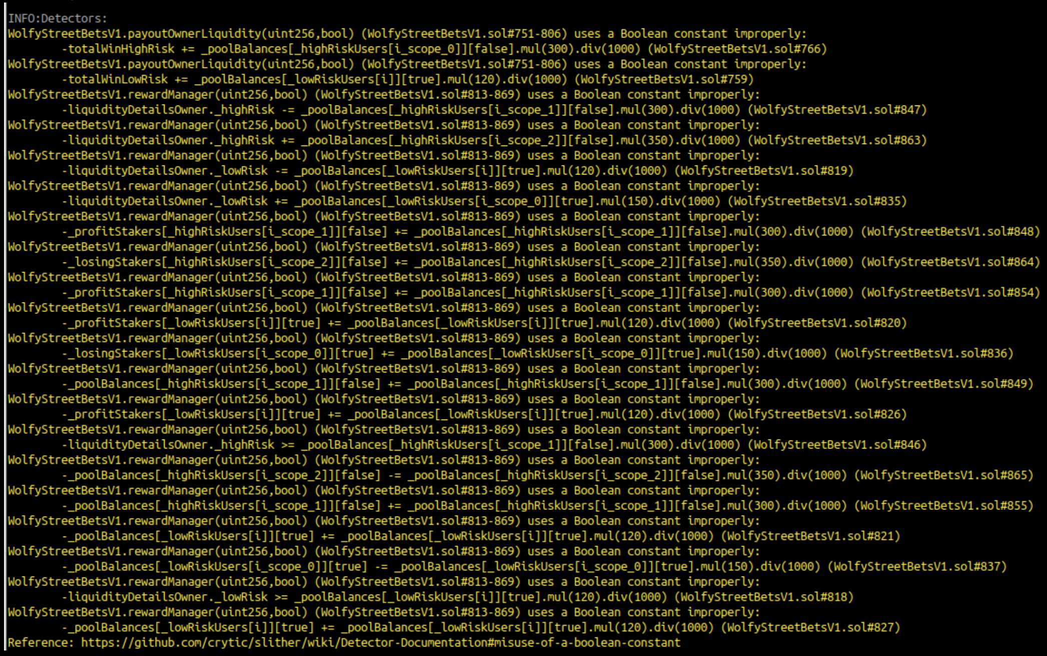
3.11 STATIC ANALYSIS REPORT

Description:

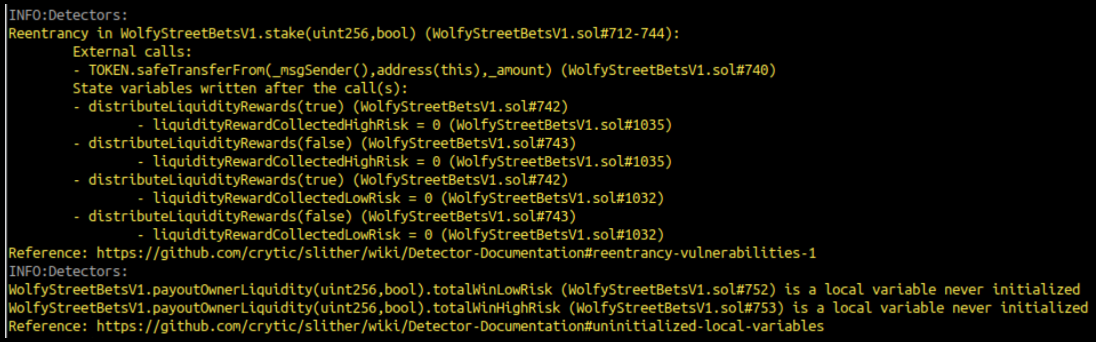
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats,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

Results:

WolfyStreetBetsv1.sol



```
INFO:Detectors:
WolfyStreetBetsV1.payoutOwnerLiquidity(uint256,bool) (WolfyStreetBetsV1.sol#751-806) uses a Boolean constant improperly:
	-totalWinHighRisk += _poolBalances[_highRiskUsers[i_scope_0]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#766)
WolfyStreetBetsV1.payoutOwnerLiquidity(uint256,bool) (WolfyStreetBetsV1.sol#751-806) uses a Boolean constant improperly:
	-totalWinLowRisk += _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#759)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-liquidityDetailsOwner._highRisk -= _poolBalances[_highRiskUsers[i_scope_1]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#847)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-liquidityDetailsOwner._highRisk += _poolBalances[_highRiskUsers[i_scope_2]][false].mul(350).div(1000) (WolfyStreetBetsV1.sol#863)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-liquidityDetailsOwner._lowRisk -= _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#819)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-liquidityDetailsOwner._lowRisk += _poolBalances[_lowRiskUsers[i_scope_0]][true].mul(150).div(1000) (WolfyStreetBetsV1.sol#835)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_profitStakers[_highRiskUsers[i_scope_1]][false] += _poolBalances[_highRiskUsers[i_scope_1]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#848)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_losingStakers[_highRiskUsers[i_scope_2]][false] += _poolBalances[_highRiskUsers[i_scope_2]][false].mul(350).div(1000) (WolfyStreetBetsV1.sol#864)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_profitStakers[_highRiskUsers[i_scope_1]][false] += _poolBalances[_highRiskUsers[i_scope_1]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#854)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_profitStakers[_lowRiskUsers[i]][true] += _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#820)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_losingStakers[_lowRiskUsers[i_scope_0]][true] += _poolBalances[_lowRiskUsers[i_scope_0]][true].mul(150).div(1000) (WolfyStreetBetsV1.sol#836)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_poolBalances[_highRiskUsers[i_scope_1]][false] += _poolBalances[_highRiskUsers[i_scope_1]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#849)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_profitStakers[_lowRiskUsers[i]][true] += _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#826)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-liquidityDetailsOwner._highRisk >= _poolBalances[_highRiskUsers[i_scope_1]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#846)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_poolBalances[_highRiskUsers[i_scope_2]][false] -= _poolBalances[_highRiskUsers[i_scope_2]][false].mul(350).div(1000) (WolfyStreetBetsV1.sol#865)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_poolBalances[_highRiskUsers[i_scope_1]][false] += _poolBalances[_highRiskUsers[i_scope_1]][false].mul(300).div(1000) (WolfyStreetBetsV1.sol#855)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_poolBalances[_lowRiskUsers[i]][true] += _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#821)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_poolBalances[_lowRiskUsers[i_scope_0]][true] -= _poolBalances[_lowRiskUsers[i_scope_0]][true].mul(150).div(1000) (WolfyStreetBetsV1.sol#837)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-liquidityDetailsOwner._lowRisk >= _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#818)
WolfyStreetBetsV1.rewardManager(uint256,bool) (WolfyStreetBetsV1.sol#813-869) uses a Boolean constant improperly:
	-_poolBalances[_lowRiskUsers[i]][true] += _poolBalances[_lowRiskUsers[i]][true].mul(120).div(1000) (WolfyStreetBetsV1.sol#827)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#misuse-of-a-boolean-constant
```



```
INFO:Detectors:
Reentrancy in WolfyStreetBetsV1.stake(uint256,bool) (WolfyStreetBetsV1.sol#712-744):
        External calls:
        - TOKEN.safeTransferFrom(_msgSender(),address(this),_amount) (WolfyStreetBetsV1.sol#740)
        State variables written after the call(s):
        - distributeLiquidityRewards(true) (WolfyStreetBetsV1.sol#742)
                - liquidityRewardCollectedHighRisk = 0 (WolfyStreetBetsV1.sol#1035)
        - distributeLiquidityRewards(false) (WolfyStreetBetsV1.sol#743)
                - liquidityRewardCollectedHighRisk = 0 (WolfyStreetBetsV1.sol#1035)
        - distributeLiquidityRewards(true) (WolfyStreetBetsV1.sol#742)
                - liquidityRewardCollectedLowRisk = 0 (WolfyStreetBetsV1.sol#1032)
        - distributeLiquidityRewards(false) (WolfyStreetBetsV1.sol#743)
                - liquidityRewardCollectedLowRisk = 0 (WolfyStreetBetsV1.sol#1032)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
WolfyStreetBetsV1.payoutOwnerLiquidity(uint256,bool).totalWinLowRisk (WolfyStreetBetsV1.sol#752) is a local variable never initialized
WolfyStreetBetsV1.payoutOwnerLiquidity(uint256,bool).totalWinHighRisk (WolfyStreetBetsV1.sol#753) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

According to the test results, most of the findings found by slither were considered as false positives. Relevant findings were reviewed by the auditors.

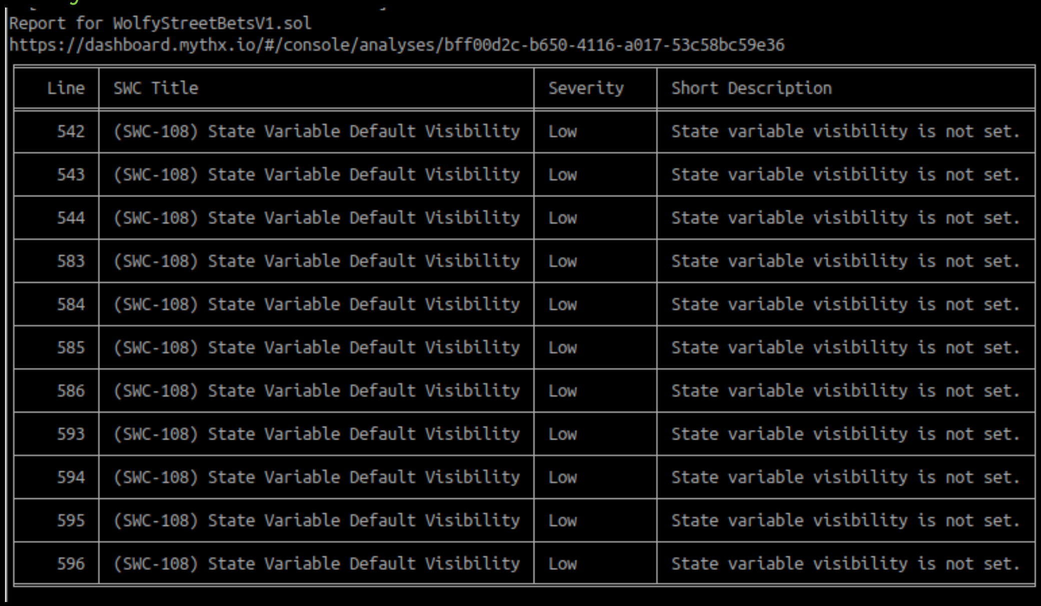
3.12 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit 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 the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Only security-related findings are shown below.

Results:

WolfyStreetBetsv1.sol



```
Report for WolfyStreetBetsV1.sol
https://dashboard.mythx.io/#/console/analyses/bff00d2c-b650-4116-a017-53c58bc59e36

| Line | SWC Title                                   | Severity | Short Description                    |
|------|---------------------------------------------|----------|--------------------------------------|
|  542 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  543 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  544 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  583 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  584 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  585 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  586 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  593 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  594 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  595 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
|  596 | (SWC-108) State Variable Default Visibility | Low      | State variable visibility is not set. |
```

THANK YOU FOR CHOOSING

// HALBORN