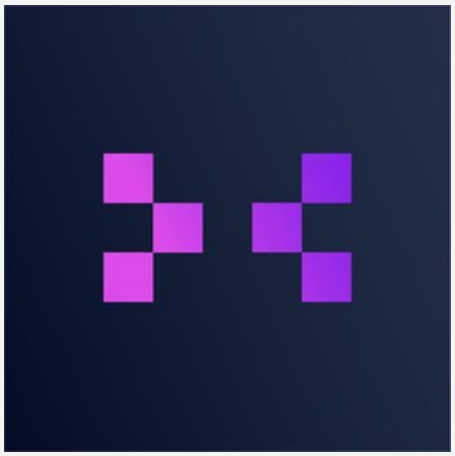
MIDGAR

SMART CONTRACT
SECURITY AUDIT OF



DEGAMING

TABLE OF CONTENTS

Project Summary

Security Firm: Midgar

Prepared By: VanGrim, EVDoc

Client Firm: First Block AB

Final Report Date -

DeGaming engaged Midgar (former Zanarkand) to review the security of its smart contracts related to the DeGaming platform. From the **26th of February to the 10th of March**, a team of two (2) auditors reviewed the source code in scope. All findings have been recorded in the following report.

Please refer to the complete audit report below for a detailed understanding of risk severity, source code vulnerability, and potential attack vectors.

Project Name	DeGaming
Language	Solidity
Codebase	https://github.com/degamingio/contracts
Commit	Initial: 9266cf23ebbe97c29eebe7514bdc5f517bcccd52 Final: c6182407755cf2135ffd555daba1dc6c9552c2e5
Audit Methodology	Static Analysis, Manual Review
Review Period	26 February - 10 March 2024
Resolved	4 April 2024

Project Overview

DeGaming introduces a pioneering decentralized gaming platform designed to transform the iGaming industry by merging the realms of licensed Web2 and unlicensed Web3 operators. This innovative platform aims to address major industry challenges, including slow innovation, questionable game fairness, high transaction costs, and centralized control, which have historically impeded the sector's growth.

Through a unique blend of blockchain technology and smart contracts, DeGaming promises to enhance transparency, fairness, and efficiency across the iGaming landscape. It offers game developers, casino operators, and investors a collaborative ecosystem where innovation is rewarded, transactions are streamlined, and game integrity is assured. By simplifying access to a wide array of games and enabling secure, low-cost payments, DeGaming sets a new standard for fairness and player trust in iGaming, positioning itself as a future leader in the digital gaming revolution.

Audit Scope

ID	File	SHA-1 Checksum
CAS	Casino.sol	**fe48369b940abf3f441686c1852f8251355eb2d1**
CNF	CoinFlip.sol	**92ca1802ef793214f55f922e802068d4190573bb**
AMG	DGAssetManager.sol	**66723da34d8ba26e13b4de0fb1157d3b71cecd04**
CAF	DGCasinoFactory.sol	**ddcb66c4a213f60b637eb9f69fc33f428c51ae54**
FMG	DGFeeManager.sol	**b5f8116bf65cb69b3b575ad7d43e418d492249af**
GAM	DGGame.sol	**f2899a0d36708063c1c550e6351c29ac008ac7ca**
GMG	DGGameManager.sol	**acd1c3fa20e1f0ec7b1920b4ac500fc2e11ebbb5**
RDM	DGRandomness.sol	**482289dd8ed1060fbfd48f6cf1003bad1939f926**

ID	File	SHA-1 Checksum
RDE	DGRandomnessDirectFunding.sol	**f0291cf3075917ad341d1c4c1553b1286e55d610**
TRF	DGTrustedForwarder.sol	**dec026a5e14cc411b920af23a789743d2e086133**
VLT	DGVault.sol	**0ade25456b16c05edfe063d68000f24733cc9390**
ADP	DGVRFAdapter.sol	**cd071b6107c7a4b687f9dfe3f140e771b70894da**
DFA	DGVRFDirectFundingAdapter.sol	**b0fdc625ce5c2f30f95c3c2a594e8a91de7e8a91**
DCE	Dice.sol	**1593afa0e5d85d8c795995271e8687a983571067**
RPS	RockPaperScissors.sol	**c880ffa99003f848b3a9e37992c2622a9a8249cc**
GLOBAL	-	**-**

Vulnerability Summary*

Vulnerability Level	Total	Acknowledged & Closed	Resolved
🔴 Critical	5	0	5
🟠 High	1	0	1
🟡 Medium	11	3	8
🟢 Low	6	0	6

**Considering the large number of critical/high vulnerabilities found during this time-boxed security review, Midgar recommends additional security testing on the codebase prior to any deployment*

Findings & Resolutions

ID	Title	Severity	Status
TRF-1	The forwarder cannot charge fee after deployment	🔴 **Critical**	**Resolved**
TRF-2	The forwarder will not be able to call the casino due to incorrect ABI encoding	🔴 **Critical**	**Resolved**
VLT-1	`GGR` considered as liquidity might disrupt the protocol	🔴 **Critical**	**Resolved**
CAS-1	A removed game stays in casinos and retains its `GAME` role	🔴 **Critical**	**Resolved**
CAS-2	An attacker could drain casinos by playing twice with different tokens	🔴 **Critical**	**Resolved**

ID	Title	Severity	Status
CAS-3	A player can play risk-free if Chainlink's response is over 30 seconds	🟠 **High**	**Resolved**
FMG-1	`claimFees()` might deplete the vault when a casino updates their fees	🟡 **Medium**	**Resolved**
FMG-2	Fees to `gameDev` will likely be truncated	🟡 **Medium**	**Acknowledged & Closed**
FMG-3	The `nullGGR()` function might erase existing GGR	🟡 **Medium**	**Acknowledged & Closed**
GMG-1	Wrong revoking role of previous game address in `updateGameAddress()`	🟡 **Medium**	**Resolved**
TRF-3	Replay attack If `requestPlay()` reverts when called by forwarder	🟡 **Medium**	**Resolved**

ID	Title	Severity	Status
TRF-4	Static `fee` variable will give different amount depending on the ERC20-token	🟡 **Medium**	**Resolved**
RDM-1	`fulfillRandomWords()` could revert under certain circumstances	🟡 **Medium**	**Acknowledged & Closed**
RDM-2	`revokeGameRole` is granting a role instead of revoking a role	🟡 **Medium**	**Resolved**
GLOBAL-1	No storage gap in upgradeable contracts can lead to storage collision	🟡 **Medium**	**Resolved**
GLOBAL-2	Incorrect branch of OZ library used in upgradeable contracts	🟡 **Medium**	**Resolved**
CAS-4	Insufficient validation could lead to game being overwritten	🟡 **Medium**	**Resolved**
RDM-3	Unused `CasinoRevenueTracker` struct	🟢 **Low**	**Resolved**
AMG-1	Fees will become stuck in the DGFeeManager if a token is removed	🟢 **Low**	**Resolved**

ID	Title	Severity	Status
CAF-1	Redundant granting of roles	🟢 **Low**	**Resolved**
GLOBAL-3	Floating pragma is not recommended in production	🟢 **Low**	**Resolved**
CNF-1	Unnecessary `_decodeParams()` function in CoinFlip.sol	🟢 **Low**	**Resolved**
CAS-5	`gameSessions[sender].wager` stored based on the user's input	🟢 **Low**	**Resolved**

TRF-1	The forwarder cannot charge fee after deployment		
Asset	DGTrustedForwarder.sol: L74		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

The DGTrustedForwarder.sol is missing to initialize the OwnableUpgradeable contract from OpenZeppelin, meaning that all functions with the modifier onlyOwner will default to address(0). This means that the setFee() function can never be accessed after deployment, leading to the fee defaulting to 0. As a consequence, the DGTrustedForwarder will never be able to take out fees. A false positive in the test (see POC) shows that the owner of the contract is never set (e.g. defaulting to address(0)). Thus, the vm.prank is impersonating address(0).

Recommendation

Add the __Ownable.init() initializer in the initialize() function.

Resolution

This issue is resolved as of commit c4dc03c2913604028591a72a635fe2bddbef39d2.

TRF-2	The forwarder will not be able to call the casino due to incorrect ABI encoding		
Asset	DGTrustedForwarder.sol:L115		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The playGame() function is ABI-encoding the _playParams in order to call and interact with the requestPlay() function in the Casino.sol contract. However, the current paramData is a mismatch to the requestPlay() in the Casino.sol. The paramData looks like the following:

```
bytes memory paramData = abi.encodeWithSignature(
"requestPlay(uint256,bytes,uint256,uint256)", _playParams.gameId, _playParams.params,
_playParams.betSize, fee);
```

Meanwhile, the requestPlay() in Casino.sol looks like this:

```
function requestPlay(uint16 _gameId, bytes memory _params, uint240 _wager, address _token, uint256
_fee)
```

The current call is missing the address _token input and in addition the unsigned integers are of different sizes which means that the function called will be non-existent and thus fail the call.

Recommendation

Correct the paramData to the following:

```
    bytes memory paramData = abi.encodeWithSignature(
    +"requestPlay(uint16,bytes,uint240,address,uint256)", _playParams.gameId, _playParams.params,
    _playParams.betSize,
    + _playParams.token, fee);
```

Resolution

This issue is resolved as of commit c6182407755cf2135ffd555daba1dc6c9552c2e5.

VLT-1		`GGR` considered as liquidity might disrupt the protocol	
Asset		DGVault.sol L125 & L262	
Status		Resolved	
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

The GGR of each casino inflates the vault's balance. However, this balance is used to calculate shares when a liquidity provider deposits funds and to determine the amount of tokens when liquidity providers withdraw shares. The vault's balance is also used to calculate the maxWager. If a significant amount of GGRs accumulates in the vault, it will distort the calculations during liquidity deposits or withdrawals and the maxWager calculation.

Recommendation

Create a variable to track amounts deposited by liquidity providers and liquidity provider's fees. Utilize this variable to determine the exchange rate and the maxWager instead of the balance of the vault.

Resolution

The issue is resolved as of commit c6182407755cf2135ffd555daba1dc6c9552c2e5

CAS-1	A removed game stays in casinos and retains its `GAME` role		
Asset	DGGameManager.sol L85		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

When a game is removed from the DeGaming game catalog in DGGameManager, it is removed from the dgGames mapping. However, the removed game retains its privileges with DGRandomness. Players would be able to continue playing this game even if it contains a vulnerability and has been removed from the DGGameManager catalog.

Recommendation

Casino.sol: Check that the game is in catalog in requestPlay() and addGame()

```
if (IDGGameManager(gameManager).dgGames(_gameId) == address(0) revert DGErrors.FORBIDDEN()
```

DGGameManager.sol: Revoke the GAME_ROLE when a game is removed from DeGaming game catalog in removeGame() function:

```
_gameAddress = dgGames[_gameId]
dgRandomness.revokeGameRole(_gameAddress)
```

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

CAS-2	An attacker could drain casinos by playing twice with different tokens		
Asset	Casino.sol L202		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

The requestPlay() function verifies the game session stage at line 198. If the stage is PAID and the session was created more than 30 seconds ago, the previous wager is sent back to the player. However, the token is instantiated at the beginning of the function (at line 180) based on the user's input.

An attacker could exploit this vulnerability by first calling requestPlay() using USDC with a wager of 100 USDC. Then, if the session has expired, they could call requestPlay() again using WBTC with a wager of 1 wei. Consequently, the casino would send them 1 WBTC (100e6 <=> 1e8).

Recommendation

Transfer back the wager to the player in the game session stage check (at line 202) based on IDGGame(games[_gameId].token)

Resolution

This issue is resolved as of commit c4dc03c2913604028591a72a635fe2bddbef39d2.

CAS-3		A player can play risk-free if Chainlink's response is over 30 seconds	
Asset		Casino.sol L229	
Status		Resolved	
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

When calling refundWager() after 30 seconds without a response from chainLink, the session's stage transitions from PAID to REFUNDED. However, there is no stage verification in the fulfillPlay() function. If Chainlink's response takes more than 30 seconds for any reason, such as network congestion, and the player receives a refund, the game will still call fulfillPlay().

As a result, the wager can be returned to the player (for a second time) in the case of a tie; alternatively, in case of a win, the player will receive both the wager and the reward.

Recommendation

Check that the stage is not REFUNDED in fulfillPlay() function.

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

FMG-1		`claimFees()` might deplete the vault when a casino updates their fees	
Asset		DGFeeManager.sol:L147	
Status		Resolved	
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description (POC)

Due to fees being claimed in both claimGameDevFees() and claimFees(), which are called by two different functions, a larger-than-intentional fee claim might be extracted from the vault.

Consider the following scenario:

- The fees are initially set at 25% each.
- GameDevFee amount increases by 25% of each wager lost.
- claimGameDevFees() is called by the game dev address
- The fees are updated: _gameDevFee to 5%, `_liquidityProviderFee` to 5% to increase `_operatorFee` to 40%, and `_deGamingFee` to 50%
- claimFees() is called, _deGamingFee added to _operatorFee represents 90% of the GGR.
- However, claimGameDevFees() has already been called, and gameDevs received an amount based on the initial fee structure.

In this scenario, the excess received by the GameDev address based on the old fee structure might be supported by the liquidity of the vault. However, the accumulation of these amounts should be distributed according to the same fee structure, without the need to draw additional the liquidity from the vault.

Recommendation

claimFees() should be called within the setCasinoFees() function, before updating the fee structure.

Resolution

This issue is resolved as of commit c4dc03c2913604028591a72a635fe2bddbef39d2.

FMG-2	Fees to `gameDev` will likely be truncated		
Asset	DGFeeManager.sol:L316		
Status	Closed		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

In the incrementGameDevFees function, the fees to the gameDev is calculated as follows:

```
int256 fees = _wager * int64(feeInfo.gameDev) / int256(DENOMINATOR);
```

If _wager * feeInfo.gameDev is non-divisible by the DENOMINATOR, fees will be truncated and left out, meaning that the gameDev might receive less than intended.

Recommendation

There are two recommendations to consider:

- First, consider adding a minimum _wager amount.
- In addition, consider adding incremental bet sizes to make sure that _wager * feeInfo.gameDev is fully divisible for every incremental bet size. E.g. fees % DENOMINATOR == 0.

Resolution

Acknowledged and closed.

FMG-3	The `nullGGR()` function might erase existing `GGR`		
Asset	DGFeeManager.sol:L200-204		
Status	Closed		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

When a user calls claimFees(), the fees are transferred to deGaming and the operator. The GGR for that casino's token will also be zeroed. However, there's a high likelihood that the fees will be truncated and rounded down to the closest integer due to how Solidity works, resulting in lost fees. The code snippet that affects the truncation is here:

```
uint256 feeToDeGaming = uint256(GGR) * feeInfo.deGaming / DENOMINATOR;
uint256 feeToGameDev = uint256(GGR) * feeInfo.gameDev / DENOMINATOR;
uint256 feeToOperator = uint256(GGR) * feeInfo.operator / DENOMINATOR;
uint256 feeToLiquidityProvider = uint256(GGR) * feeInfo.liquidityProvider / DENOMINATOR;
```

Recommendation

Consider checking that the fees are fully divisible with DENOMINATOR. E.g. feeToDegaming % DENOMINATOR == 0. This requires that the product of GGR * feeInfo is a multiple of the DENOMINATOR.

Resolution

Acknowledged and closed.

GMG-1		Wrong revoking role of previous game address in `updateGameAddress()`	
Asset		DGGameManager.sol L105	
Status		Resolved	
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The revocation of the GAME_ROLE from the previous game contract within the DGRandomness contract is not performed correctly in the updateGameAddress function.

The new address, which was supposed to update the game's address, has its role revoked, whereas the old address should lose its role.

```
dgRandomness.revokeGameRole(_gameAddress);
```

Recommendation

Revoke the GAME_ROLE from the previous game contract:

```
previousGameAddress = dgGames[_gameId];
dgRandomness.revokeGameRole(previousGameAddress);
```

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

TRF-3		Replay attack If `requestPlay()` reverts when called by forwarder	
Asset		DGTrustedForwarder L 128	
Status		Resolved	
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

When a user uses the forwarder, if the call to requestPlay() is unsuccessful, playGame() will revert, and lastNoncePerSession won’t be incremented. requestPlay() can revert for various reasons, such as if the user does not have enough balance at this time or if the wager is too high compared to the liquidity of the vault. Even a reverted transaction will be shown on-chain and can be replayed under the right circumstances. Anyone could resend the transaction with the same arguments if lastNoncePerSession hasn't been incremented in the meantime. If the conditions are met at that time, the transaction could proceed at the player's expense.

Recommendation

Do not revert if the transaction L124 fails, allowing the nonce to be incremented.

Resolution

This issue is resolved as of commit c4dc03c2913604028591a72a635fe2bddbef39d2.

TRF-4	Static `fee` variable will give different amount depending on the ERC20-token		
Asset	DGTrustedForwarder.sol:L21		
Status	Resolved		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The fee variable in the DGTrustedForwarder.sol is static and doesn't consider different ERC20 tokens that might have different decimals. For example, the USDC and USDT have 6 decimals, while the DAI token has 18. If the fee is set while considering only USDC and USDT, then the fee will have a less than expected amount when the forwarder is used with a token that has 18 decimals.

Recommendation

Make the fee variable dynamic and dependent on the token being used.

Resolution

This issue is resolved as of commit c4dc03c2913604028591a72a635fe2bddbef39d2.

RDM-1	`fulfillRandomWords()` could revert under certain circumstances		
Asset	DGRandomness.sol L103		
Status	Closed		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

According to the documentation of Chainlink VRF, the contract logic should not revert when fulfillRandomWords is called. In DGRandomness.sol, fulfillRandomness() calls evaluatePlayResults, which might revert. This might occur if:

- Vault's balance is insufficient, and transferRewards() reverts.
- Casino has been blocked, resulting in a revert of incrementGameDevFee() or decrementGameDevFee()

Recommendation

Check the casino's status in requestPlay().
Consider caching the randomness received as recommended by Chainlink.

Resolution

Acknowledged and closed.

RDM-2	`revokeGameRole` is granting a role instead of revoking a role		
Asset	DGRandomness.sol:L91, DGRandomnessDirectFunding.sol:L74		
Status	Resolved		
Rating	Severity: Medium	Impact: Low	Likelihood: High

Description

The function revokeGameRole is granting the GAME_ROLE to the _gameContract instead of revoking it.

Recommendation

Replace _grantRole() function with the _revokeRole() function.

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

GLOBAL-1	No storage gap in upgradeable contracts can lead to storage collision		
Asset	DGTrustedForwarder.sol, Casino.sol		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The DGTrusterForwarder.sol and the Casino.sol are currently missing a gap variable, as the other contracts have. Thus, during upgrades, these contracts risk having variables that cause their child contracts to be overwritten.

Recommendation

Consider adding a uint256[30] __gap; to the contracts described above.

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

GLOBAL-2	Incorrect branch of OZ library used in upgradeable contracts		
Asset	Casino.sol L6 , DGFeeManager.sol L6 , DGVault.sol L6		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Casino.sol, DGFeeManager.sol, and DGVault.sol use the safeERC20 OZ library. From the README file of the upgrades safe library: “You must use this package and not @openzeppelin/contracts if you are writing upgradeable contracts.” Using the upgrades safe library, in this case, will ensure the inheritance from Initializable and the other contracts is always linearized as expected by the compiler

Recommendation

Use safeERC20Upgradeable library instead.

```
import {SafeERC20Upgradeable} from
"@openzeppelin/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol";
```

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

CAS-4	Insufficient validation could lead to game being overwritten		
Asset	Casino.sol:L308		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

When an admin or operator adds a game through the addGame() function, there's no validation checking to see if the _gameId is already in use. This means that the admin could mistakenly overwrite an existing game.

Recommendation

Consider adding the following validation and add the error type DGErrors.GAME_EXISTS():

```
if(games[_gameId] != address(0)) revert DGErrors.GAME_EXISTS()
```

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

RDM-3	Unused `CasinoRevenueTracker` struct		
Asset	DGDataTypes.sol:L40		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The CasinoRevenueTracker struct is declared in DGDataType.sol and is not used anywhere. This struct is supposed to track the profit, losses, and the net profit of casinos

Recommendation

Update CasinoRevenueTracker.profit, CasinoRevenueTracker.loss and CasinoRevenueTracker.netProfit accordingly in fulfillPlay()

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

AMG-1		Fees will become stuck in the DGFeeManager if a token is removed	
Asset		DGAssetManager.sol L74	
Status		Resolved	
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

If a token is removed from the catalog in DGAssetManager.sol, it will be impossible to claim the fees. Both claimFees() and claimDevFees() will revert due to the following check:

```
if (!dgAssetManager.checkToken(_token)) revert DGErrors.TOKEN_NOT_IN_CATALOG();
```

Recommendation

Call claimFees() from within removeToken(). Consider adding an onlyOwner withdraw() function in DGFeeManager.sol to withdraw the remaining tokens

Resolution

This issue is resolved as of commit c4dc03c2913604028591a72a635fe2bddbef39d2.

CAF-1	Redundant granting of roles		
Asset	CasinoFactory.sol:L135		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

We are granting the same roles in both Casino.sol and CasinoFactory.sol for the Casino contract, which is reduntant.

Recommendation

Consider removing the following code since the roles are already granted in Casino.sol.

```
newCasino.grantRole(newCasino.DEGAMING_ADMIN_ROLE(), dgAdmin);
newCasino.grantRole(newCasino.FEEMANAGER_ADMIN_ROLE(), dgFeeManager);
newCasino.grantRole(DEFAULT_ADMIN_ROLE, _operator);
```

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

GLOBAL-3	Floating pragma is not recommended in production		
Asset	-		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Contracts should be locked to a fixed pragma version to not accidentally deploy a pragma version with unknown vulnerabilities.

Recommendation

Consider fixing the pragma version for all contracts.

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

CNF-1	Unnecessary `_decodeParams()` function in CoinFlip.sol		
Asset	CoinFlip.sol		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Given that the play() function in CoinFlip.sol doesn't decode the _params, there's no need for the decodeParams() function.

Recommendation

Consider removing _decodeParams() function in CoinFlip.sol

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

CNF-1		`gameSessions[sender].wager` stored based on the user's input	
Asset		Casino.sol:L211	
Status		Resolved	
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Storing the wager in the gameSession mapping according to the user's input is unsafe. It is common practice to store the amount sent by users according to changes in the balance of the contract. This measure could help prevent exploits arising from uncommon token behaviors, like those observed with cUSDCv3, where a transfer/transferFrom of type(uint256).max does not revert the transaction if the user lacks sufficient balance; instead, the user's entire balance is sent. Similar risks may emerge with new behaviors introduced in upgradeable tokens, such as USDC or USDT.

Recommendation

- Store the balance of the casino before the safeTransferFrom.
- Store the balance of the casino after the safeTransferFrom.
- Define the wager based on the difference.

```
uint256 wager = balanceAfter - balanceBefore;
```

Resolution

This issue is resolved as of commit b6c5e3853a5ea46cd7f38f013a3a5884ec38d1a0.

Appendix

Tests

TRF - 1

```
[PASS] test_OnlyOwnerCanSetFee() (gas: 48136)
Traces:
  [48136] forwarderTest::test_OnlyOwnerCanSetFee()
    ├─ [9490] forwarder::owner() [staticcall]
    │   ├─ [2354] DGTrustedForwarder::owner() [delegatecall]
    │   │   └─ ← 0x0000000000000000000000000000000000000000
    │   └─ ← 0x0000000000000000000000000000000000000000
    ├─ [0] VM::startPrank(0x0000000000000000000000000000000000000000)
    │   └─ ← ()
    ├─ [23192] forwarder::setFee(100)
    │   ├─ [22556] DGTrustedForwarder::setFee(100) [delegatecall]
    │   │   └─ ← ()
    │   └─ ← ()
    ├─ [986] forwarder::fee() [staticcall]
    │   ├─ [350] DGTrustedForwarder::fee() [delegatecall]
    │   │   └─ ← 100
    │   └─ ← 100
    ├─ [0] VM::stopPrank()
    │   └─ ← ()
```

Vulnerability Classification

The risk matrix below has been used for rating the vulnerabilities in this report. The full details of the interpretation of the below can be seen here.

High Impact	**Medium**	**High**	**Critical**
Medium Impact	**Low**	**Medium**	**High**
Low Impact	**Low**	**Low**	**Medium**
	Low Likelihood	**Medium Likelihood**	**High Likelihood**

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by aspiring auditors.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Midgar to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Midgar’s position is that each company and individual are responsible for their own due diligence and continuous security. Midgar’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Midgar are subject to dependencies and are under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access and depend upon multiple layers of third parties.

Notice that smart contracts deployed on the blockchain are not resistant to internal/external exploits. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Midgar does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Midgar

Midgar is a team of security reviewers passionate about delivering comprehensive web3 security reviews and audits. In the intricate landscape of web3, maintaining robust security is paramount to ensure platform reliability and user trust. Our meticulous approach identifies and mitigates vulnerabilities, safeguarding your digital assets and operations. With an ever-evolving digital space, continuous security oversight becomes not just a recommendation but a necessity. By choosing Midgar, clients align themselves with a commitment to enduring excellence and proactive protection in the web3 domain.

To book an audit, message https://t.me/vangrim1.

MIDGAR