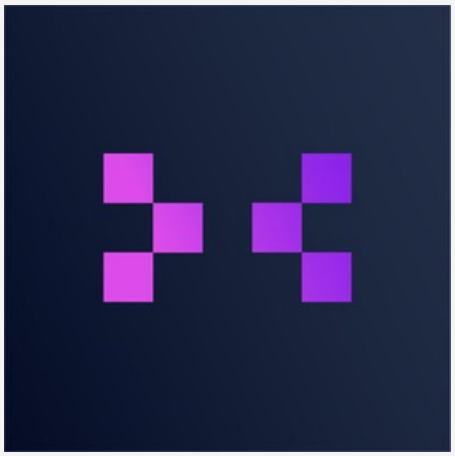
MIDGAR

SMART CONTRACT
SECURITY AUDIT OF



DEGAMING

TABLE OF CONTENTS

Project Summary

Security Firm: Midgar Pte. Ltd.

Prepared By: VanGrim, EVDoc

Client Firm: First Block AB

Final Report Date: 30th September 2024

DeGaming engaged Midgar (former Zanarkand) to review the security of its smart contracts related to the DeGaming platform. From the **10th of April to the 26th of April**, 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: ba9e1be632e2aba8af2e203ec09dca72fffef362 Final: 4827d99b184def600bc793a938c7f4bbdcf52438
Audit Methodology	Static Analysis, Manual Review, Fuzz Testing, Invariant Testing
Review Period	10 April - 26 April 2024
Resolved	8 May 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	**cfda1e1ad2bac5afd753c7d5ada775ea56a6d8f1**
CNF	CoinFlip.sol	**72b3d64e045431baa9615039a56b5335dbe9b5c5**
AMG	DGAssetManager.sol	**4e31597f547374098d9f7a2c9b72d76ee0dba175**
CAF	DGCasinoFactory.sol	**3c35dc868ad6cbc56548f210e7610ef621ea4b84**
FMG	DGFeeManager.sol	**092bb4a84a831f1079047d445d7749a075e57466**
GAM	DGGame.sol	**4437456faa23d21fbe10c7895b2fae1cf3a60d01**
GMG	DGGameManager.sol	**7e87177e940f7775eeb8bc8dc4c4621aee9e38fd**
RDM	DGRandomness.sol	**d23f2b6edf73265dc2240fec5262c10fbfa18b57**

Audit Scope

ID	File	SHA-1 Checksum
TRF	DGTrustedForwarder.sol	**a2cf941326f858b820dd06452c4a5c4886529f40**
VLT	DGVault.sol	**6a4223d757a8e9ad9d05070218c7ecbb4adbf682**
ADP	DGVRFAdapter.sol	**26e524960880a047964799c66ad79dcfe7298fa6**
DCE	Dice.sol	**c2a891f032ee6d55a25535bfe1b2a3da2ee2064e**
RPS	RockPaperScissors.sol	**176f1f321d722f29cb14e862ea9f7ca5e61967ac**
GLOBAL	-	**-**

Vulnerability Summary

Vulnerability Level	Total	Acknowledged	Resolved
Critical	5	0	5
High	1	0	1
Medium	2	0	2
Low	1	0	1
Informational	1	0	1

Findings & Resolutions

ID	Title	Severity	Status
DGV-1	The `WHITELISTED_LP` variable is not properly initialised	🔴 **Critical**	**Resolved**
DGV-2	An LP always benefits from frontrunning the `fulfillplay()` function	🔴 **Critical**	**Resolved**
DGV-3	A large LP can DoS `fulfillPlay()` function	🔴 **Critical**	**Resolved**
DGF-1	`claimGameDevFees()` and `claimFees()` might DoS due to arithmetic underflow	🔴 **Critical**	**Resolved**
DGF-2	`claimGameDevFees()` reduces liquidity and negatively affects the exchange rate of shares	🔴 **Critical**	**Resolved**
DGF-3	LPs might lose liquidity when calling `claimFees()`	🟠 **High**	**Resolved**
CSN-1	A chain-reorg can cause the function `fulfillPlay()` to provide a non-existent game session	🟡 **Medium**	**Resolved**
DGV-4	`DGVault` vulnerable to inflation attacks	🟡 **Medium**	**Resolved**
DGG-1	NATSPEC error for `maxWager()` function across three games	🟢 **Low**	**Resolved**
GLOBAL-1	Implementing the `PausableUpgradeable` library	🔵 **Informational**	**Resolved**

DGV-1	The `WHITELISTED_LP` variable is not properly initialised		
Asset	DGVault.sol: L28		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

In the `DGVault.sol` contract, the `WHITELISTED_LP` variable is not properly initialised. Due to the proxy upgradeable pattern, the implementation contract and the proxy contract storing the variables, are initialised separately.
In this case, the `onlyRole` modifier in the `depositFunds()` function in the implementation contract (`DGVault.sol`) is called before the state variables have been initialised in the proxy contract.
As shown in the POC below, this leads to the `depositFunds()` function only being accessible to the default value of `address(0)` instead of whitelisted LPs with the `keccak256("WHITELISTED_LP")` role.
As a result, even if whitelisted, LPs will not be able to deposit any funds.

Recommendation

Change the `WHITELISTED_LP` initialisation to the following to make sure the role is initialised during compile time:

```
bytes32 public constant WHITELISTED_LP = keccak256("WHITELISTED_LP");
```

Resolution

This issue is resolved as of commit 68e6f70bbca3fc7ddb64923a1e804aae31a23beb

DGV-2		An LP always benefits from frontrunning the `fulfillplay()` function	
Asset		Casino.sol: L229	
Status		Resolved	
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

Liquidity Providers (LPs) can frontrun any call to `fulfillPlay()` and, in a risk-free manner, withdraw their funds while securing rewards. Consider the following three scenarios where an LP can frontrun the game (without knowing the outcome beforehand) to either take profit or protect their funds:

1. When a player wins and `fulfillPlay()` is called: As demonstrated in the Proof of Concept (POC), an LP can frontrun this function by withdrawing their funds, thus safeguarding their investment. They can also immediately call `depositFunds()` to acquire more shares for a lower token allocation.
2. When a player ties: In this scenario, the player could backrun `fulfillPlay()` to redeposit their withdrawn funds, allowing them to continue participating as an LP without incurring losses.
3. When a player loses and `fulfillPlay()` is called: The Gross Gaming Revenue (GGR) increases, but since `claimFees()` has not yet been called, the liquidity remains unchanged. In this case, the LP can backrun the `fulfillPlay()` function to redeposit their funds, and then immediately call `claimFees()` to realize an instant profit.

Recommendation

Consider increasing liquidity for the LPs if _multiplier == 0 instead of solely within the `claimFees()` function. Alternatively, calling the `claimFees()` function directly to immediately increase liquidity and distribute the remaining fees. Additionally, introducing a two-step withdrawal process could prevent immediate withdrawals, adding an extra layer of security.

Resolution

This issue is resolved as of commit bd2ed3d034e4f720456183d28caefa64ecd608ec

DGV-3	A large LP can DoS `fulfillPlay()` function		
Asset	DGVault.sol: L156		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

Nothing prevents LPs from withdrawing all the liquidity, which could lead to a DoS of `fulfillPlay()` and prevent players from receiving their rewards in case of a win, forcing them to request a refund.

Recommendation

Do not allow LPs to withdraw all the liquidity provided, ensure that the equivalent of the potential rewards for ongoing sessions remains in the vault.
This example has not been tested and should only provide guidance.
Considering that the games vary, with some having a `WIN_MULTIPLIER` (coinFlip and RPS) while others have a multiplier chosen by the player (Dice), consider implementing a new variable in the games:

```
uint256 public maxWinMultiplier
```

This will enable implementing a check in `requestPlay()` to extract the max win multiplier:

```
uint256 maxWinMultiplier = game.maxWinMultiplier();
```

Then calculate the maximum potential payout, and get the maximum potential reward to add it to a sum of potential rewards variable in dgVault.sol. This mechanism requires creating two new functions in dgVault.sol: incrementSumOfPotentialRewards() and decrementSumOfPotentialRewards():

```
uint256 maxPayout = wager * (maxWinMultiplier  - DENOMINATOR) / DENOMINATOR;
uint256 maxReward = maxPayout - wager;
vault.incrementSumOfPotentialRewards(maxReward);
```

Update `sumOfPotentialRewards` in `fulfillPlay()` function:

```
uint256 maxPayout = wager * (maxWinMultiplier  - DENOMINATOR) / DENOMINATOR;
uint256 maxReward = maxPayout - wager;
vault.decrementSumOfPotentialRewards(maxReward);
```

Check that enough tokens remain in the vault when an LP calls `_withdraw()`

```
uint256 liquidity = liquidity[_token];
uint256 availableLiquidity = liquidity - sumPotentialRewards;
if (amount > availableLiquidity) revert DGErrors.NOT_ENOUGH_LIQUIDITY_FOR_TOKEN();
```

Resolution

This issue is resolved as of commit c7f38bdf9057401d9861f014a2fb2622b0a9438b

DGF-1		`claimGameDevFees()` and `claimFees()` might DoS due to arithmetic underflow	
Asset		DGFeeManager.sol: L204, L376	
Status		Resolved	
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

In `DGFeeManager.sol`, both `claimFees()` and `claimGameDevFees()` call `transferReward()` in `DGVault.sol`. `transferReward()` reduces the liquidity by `uint256(GGR) - feeToGameDev - feeToLiquidityProvider` in the case of `claimFees()`, and by the claimable amount by gameDev in the case of `claimGameDevFees()`.

```
// Decrement liquidity
liquidity[_token] -= _amount;
```

However, `liquidity[_token]` can be equal to zero, leading to an arithmetic underflow and thus causing a DoS in the functions `claimFees()` and `claimGameDevFees()`.

Recommendation

Do not call `transferReward()` from `DGFeeManager` because it reduces liquidity by an amount that had been sent to the `DGVault` by `Casino` using `safeTransfer()`. The amount sent from `Casino` remains in DGVault's balance and is not part of the liquidity.

Resolution

This issue is resolved as of commit 2a04bd414686ed59ff935af4de7b9dfa222c3ffa

DGF-2		`claimGameDevFees()` reduces liquidity and negatively affects the exchange rate of shares	
Asset		DGFeeManager.sol: L359	
Status		Resolved	
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

Whenever `claimGameDevFees()` or `claimGameDevFeesAdmin` is called, the liquidity in `DGVault` is reduced by the amount claimed by the `gameDev`. Consequently, the exchange rate of shares is negatively impacted.
The exchange rate should not be affected by these fees, only player `reward` should negatively impact shares. Thus, LPs bear the risk of both player winnings and `gameDev` fee claims.
Therefore, shares would continuously lose value when `claimGameDevFees()` or `claimGameDevFeesAdmin` is called.

Recommendation

If gameDevs are not supposed to be compensated by LPs, do not deduct gameDevFees from the liquidity when `_claimGameDevFees()` is called.

Resolution

This issue is resolved as of commit 977c983b424d22112cd7ff15e1f7a4a9379280f6

DGF-3	LPs might lose liquidity when calling `claimFees()`		
Asset	DGFeeManager.sol: L204		
Status	Resolved		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description (POC)

In the `DGFeeManager.sol`, there are certain instances where the `claimFees()` function doesn't increment liquidity for LPs in line with the `liquidityProviderFee`. This is due to the function wrongly utilizing DGVault's `transferReward()` function which decrements liquidity before incrementing it. As a result, the liquidity variable will fluctuate in accordance with the current fee structure.
This means that there are instances where the liquidity providers actually lose liquidity when calling `claimFees()`.

Recommendation

Create a new function that sends tokens from the vault to the `DGFeeManager.sol` contract that replaces the `transferReward()` and that doesn't decrement liquidity from the liquidity providers.

Resolution

This issue is resolved as of commit 2a04bd414686ed59ff935af4de7b9dfa222c3ffa

CSN-1	A chain-reorg can cause the function `fulfillPlay()` to provide a non-existent game session		
Asset	DGRandomness.sol: L71		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

In `DGRandomness.sol`, the function `requestRandomNumbers()` is calling the Chainlink VRF Adapter with a set number of three (3) `_requestConfirmations` which means that VRF waits for three (3) confirmations before fulfilling the request.
However, in certain blockchains such as Polygon and Cronos - chain reorganizations happen several times a day with reorganization depth often greater than three (3) as can be seen here: https://polygonscan.com/blocks_forked.
This means that a randomness result with a certain outcome e.g. win for a player could have a changed outcome since the request of randomness from VRF is moved to a different block.

Recommendation

Consider using a larger number for `_requestConfirmations` if the protocol will be deployed on Polygon or Cronos.

Resolution

This issue is resolved as of commit 68e6f70bbca3fc7ddb64923a1e804aae31a23beb

DGV-4		`DGVault` vulnerable to inflation attacks	
Asset		DGVault.sol: L95	
Status		Resolved	
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description (POC)

The amount of shares issued is recorded in `DGVault` contract, and the assets it holds are recorded in `liquidity` mapping. This liquidity can be increased by anyone who calls `claimFees()`, this could be likened to a donation in an inflation attack. Under certain conditions, a malicious LP could take advantage of this.
Consider the following scenario:

1. An LP deposits into the vault. Players play and lose more than they win, increasing the casino's `GGR`
2. The LP withdraws all its tokens from the vault
3. A malicious LP deposits one wei into the vault and retrieves one share
4. The malicious LP calls the `claimFees()` function to increase the vault's liquidity before the next deposit into the vault.
5. A new LP deposits into the vault but will receive no shares in return if its deposit is less than `liquidity[token]`.
6. The malicious LP can then withdraw all the liquidity

Recommendation

Consider implementing a mechanism similar to Uniswap V2

Resolution

This issue is resolved as of commit 828c78c63b555601d48ec36c499433b69ff640cd

DGG-1	NATSPEC error for `maxWager()` function across three games		
Asset	CoinFlip.sol: L170; RockPaperScissors.sol: L179; Dice.sol: L207		
Status	Resolved		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The notice in the NATSPEC indicates that the function `getMaxWager()` returns a boolean, whereas it returns the `maxWager`.

Recommendation

Change the notice to `Returns the maximum wager for the given bet`

Resolution

This issue is resolved as of commit 68e6f70bbca3fc7ddb64923a1e804aae31a23beb

GLOBAL-1		Implementing the `PausableUpgradeable` library	
Asset			
Status		Resolved	
Rating	Severity: Informational		

Description

The `PausableUpgradeable` library is an Open Zeppelin library (https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/utils/PausableUpgradeable.sol) which allows a contract to enforce an emergency stop mechanism in case of malicious activity. If that happens, the owner's of the Pausable contract is able to, through a boolean value, stop any activity with a certain function.

Recommendation

Consider implementing the `PausableUpgradeable` contract if deemed necessary.

Resolution

This issue is resolved as of commit 665d4fd7a985c970c76fe9d9631b2fac5fc7ad77

Appendix

Test Suites: Invariants and Properties

During the security review, Midgar conducted an extensive series of tests on the client's codebase using the Echidna testing tool. The focus was on fuzzing and invariants testing to verify that the properties and invariants within the codebase remained robust under a variety of randomized conditions. This rigorous testing approach was designed to ensure the codebase's resilience and reliability by simulating a wide range of inputs and scenarios.

Over the course of the review, Echidna performed 1,000,000 test runs, systematically exploring potential vulnerabilities and ensuring that the codebase adhered to its specified behaviors.

Casino.sol

ID	Description	Run Count	Passed
CSN-01	Casino's balance is greater than or equal to the sum of wagers	**1,000,000+**	✅
CSN-02	Vault’s liquidity is greater than or equal to the sum of rewards	**1,000,000+**	❌
CSN-03	After executing requestPlay(), the casino's balance increases by the amount wagered	**1,000,000+**	✅
CSN-04	After executing requestPlay(), the player's balance decrease by the amount wagered	**1,000,000+**	✅
CSN-05	After executing requestPlay(), GameSessionStage is set to PAID	**1,000,000+**	✅
CSN-06	After executing refundWager(), GameSessionStage is set to REFUNDED	**1,000,000+**	✅
CSN-07	After executing fulfillPlay(), GameSessionStage is set to COMPLETED	**1,000,000+**	✅
CSN-08	After executing fulfillPlay() and player losses, the vault's balance increases by the amount wagered.	**1,000,000+**	✅
CSN-09	After executing fulfillPlay() and player losses, GameDevFees increase	**1,000,000+**	✅

ID	Description	Run Count	Passed
CSN-10	After executing fulfillPlay() and the player wins, the player's balance increases by the amount wagered plus the reward	**1,000,000+**	❌
CSN-11	After executing fulfillPlay() and the player wins, the vault's balance decreases by the reward amount	**1,000,000+**	❌
CSN-12	After executing fulfillPlay() and the player wins, GameDevFees decreases	**1,000,000+**	✅

DGFeeManager.sol

ID	Description	Run Count	Passed
DFM-01	After executing claimFees(), DeGaming's balance increases	**1,000,000+**	✅
DFM-02	After executing claimFees(), operator's balance increases	**1,000,000+**	✅
DFM-03	After executing claimFees(), liquidity increases	**1,000,000+**	❌
DFM-04	After executing claimFees(), GGR is nulled out	**1,000,000+**	✅
DFM-05	After executing claimGameDevFees(), GameDev's balance increases by claimable amount	**1,000,000+**	❌
DFM-06	After executing claimGameDevFees(), claimable amount is set to zero	**1,000,000+**	✅

DGVault.sol

ID	Description	Run Count	Passed
VLT-01	The total supply of shares equals the sum of shares minted.	**1,000,000+**	✅
VLT-02	The balance of the vault is equal or greater than the liquidity.	**1,000,000+**	✅
VLT-03	When calling `depositFunds()`, the user's shares before the deposited amount and the shares minted equals the user shares after.	**1,000,000+**	✅
VLT-04	When calling `depositFunds()`, the total supply of shares before the deposit and the shares minted equals the total supply of shares after.	**1,000,000+**	✅
VLT-05	When calling `depositFunds()`, and the deposited amount is more than 0, the minted shares is also more than 0.	**1,000,000+**	✅
VLT-06	When calling `depositFunds()`, and the deposited amount is 0, the minted shares is also 0.	**1,000,000+**	✅
VLT-07	When a user withdraws funds, the vault balance before the withdrawal subtracted by the withdrawn amount equals the vault balance after the withdrawal.	**1,000,000+**	✅
VLT-08	When a user withdraws funds, the total shares supply before the withdrawal subtracted by the burnt shares equals the total shares supply after the withdrawal.	**1,000,000+**	✅
VLT-09	When a user withdraws funds, the liquidity before the withdrawal subtracted by the withdrawn amount equals the liquidity after the withdrawal	**1,000,000+**	✅

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 a security review, message https://t.me/vangrim1.

MIDGAR