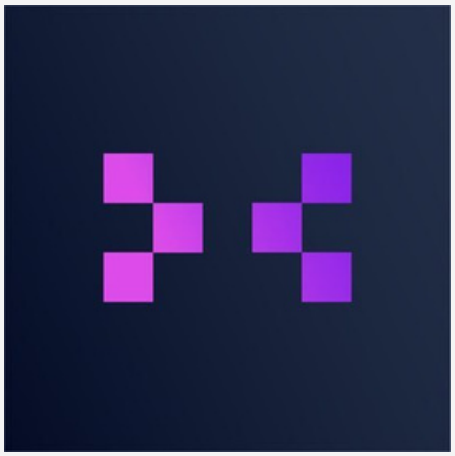
MIDGAR

SMART CONTRACT
SECURITY AUDIT OF



DEGAMING

BANKROLL CONTRACTS

TABLE OF CONTENTS

Project Summary

Security Firm: Midgar

Prepared By: VanGrim, EVDoc

Client Firm: First Block AB

Final Report Date: 16th April 2024

DeGaming engaged Midgar (former Zanarkand) to review the security of its smart contracts related to the DeGaming platform. From the **25th of Match to the 3rd 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/bankroll-contracts
Commit	Initial: f935e5e42f9b84a00a35d8171634f475b988fec5 Final: 3b4733e68d83584b93b49b73757e6bf4da0d2d2a
Audit Methodology	Static Analysis, Manual Review, Fuzz Testing
Review Period	25 March - 3 April 2024
Resolved	16th 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
BKR	Bankroll.sol	**292f81d8e1014e5c1e70286 4bf89188ac0538b03**
DBF	DGBankrollFactory.sol	**2540910ce9fed9f3f421c61 9b0c627637903b84f**
DBM	DGBankrollManager.sol	**562d3f3cda2d850a9cf2ca0 d04bd55598c3e419d**
DGE	DGEscrow.sol	**f259ab067a0010e1e5a1a65 b84e5be069f826a28**
GLOBAL	-	**-**

Vulnerability Summary*

Vulnerability Level	Total	Acknowledged	Resolved
Critical	2	2	2
High	6	6	6
Medium	6	6	4
Low	3	3	3
Informational	3	3	1

**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
BKR-1	Bankroll is vulnerable to inflation attack	🔴 **Critical**	**Resolved**
BKR-2	Potential DoS in `depositFunds()`	🔴 **Critical**	**Resolved**
BKR-3	A player can always avoid sending funds to the Escrow	🟠 **High**	**Resolved**
BKR-4	Lack of a way to preview mint for LPs	🟠 **High**	**Resolved**
BKR-5	Lack of a way to preview redeem for LPs	🟠 **High**	**Resolved**
BKR-6	Low `withdrawalDelay` and `withdrawalEventPeriod` enable front-running of `debit()`	🟠 **High**	**Resolved**
DBM-1	LP fees are sandiwchable	🟠 **High**	**Resolved**
DGE-1	The `revertFunds()` function will revert when using USDT	🟠 **High**	**Resolved**
DGE-2	Risk of ID collision	🟡 **Medium**	**Resolved**
BKR-7	Missing `gap` for contract upgrades	🟡 **Medium**	**Acknowledged & closed**
BKR-8	`ReentrancyUpgradeble` is not initialised	🟡 **Medium**	**Resolved**

Findings & Resolutions

ID	Title	Severity	Status
BKR-9	Player loses part of their debit when the `_amount` is greater than `maxRisk`	🟡 **Medium**	**Acknowledged & closed**
BKR-10	Missing validation checks could DoS withdrawals	🟡 **Medium**	**Resolved**
DGF-1	Incorrect branch of OZ library used in upgradeable contract	🟡 **Medium**	**Resolved**
BKR-11	Calling `debit()` on a bankroll with a `maxRisk` of 0 has no effect	🟢 **Low**	**Resolved**
DBM-2	Excessively low `ggrOf` values will not generate any fees	🟢 **Low**	**Resolved**
DGE-3	Players have no way of knowing the `eventPeriod` in `DGEscrow`	🟢 **Low**	**Resolved**
BKR-12	Not possible to debit Account Abstraction wallets	🔵 **Informational**	**Resolved**
GLOBAL-1	Contract size check vulnerability	🔵 **Informational**	**Acknowledged & closed**
GLOBAL-2	Following the ERC-4626 standard	🔵 **Informational**	**Acknowledged & closed**

BKR-1	Bankroll is vulnerable to inflation attack		
Asset	Bankroll.sol: L221		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

The amount of shares issued is recorded in the vault contract, but the assets it holds are not. These are recorded as the balance of the vault contract in the asset contract. This balance can be altered by anyone who transfers additional tokens to the vault's address, an action known as a donation. When the vault unexpectedly receives a donation, the value of each existing share increases.

Consider the following scenario:
* Attacker back-runs creation of a bankroll and deposits 1 wei , he gets back 1 share
* Attacker front-runs victim's deposit and transfers 200 USDC to contract
* Victim deposits 200 USDC and gets 0 shares (`1 * 200e6 / (200e6 + 1) == 0 `)
* Attacker burns their share and gets all the money

Recommendation

Mint dead shares in the constructor or “steal” a minimum amount of the first deposit and send the LP tokens to a dead address, as Uniswap V2 does.

```
constructor() {
        _disableInitializers();
        _mint(address(this), 1000);
    }
```

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-2	Potential DoS in `depositFunds()`		
Asset	Bankroll.sol: L221		
Status	Resolved		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description (POC)

The `depositFunds()` function is vulnerable to DoS under specific conditions, hindering LPs from depositing funds and minting shares. This situation may arise when the `totalSupply` is equal to or greater than 1 and a call to the `debit()` function results in zero liquidity.
Consequently, an error occurs due to division by zero.

```
if (totalSupply < 1) {
     shares = amount;
} else {
      shares = (amount * totalSupply) / liq;
}
```

Recommendation

Consider not deploying a bankroll with `maxRisk` set to 100%.
Consider prohibiting minting if `liquidity` is equal to zero and `totalSupply` is greater than 0 by adding a check in the `depositFunds()` function.

```
if (totalSupply > 0 && liq == 0) {
    revert DGErrors.NOT_ENOUGH_LIQUIDITY
}
```

Or set the exchange rate to one in this case

```
// calculate the amount of shares to mint
uint256 shares;
if (totalSupply < 1 || totalSupply > 0 && liq == 0) {
    shares = amount;
} else {
        shares = (amount * totalSupply) / liq;
}
```

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-3	A player can always avoid sending funds to the Escrow		
Asset	Bankroll.sol: L221		
Status	Resolved		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description (POC)

The purpose of the DGEscrow is to add a security measure in case of suspicious activities such as fraud, so that any amount sent from the contract will first be sent to the escrow if it exceeds the escrow threshold.

However, due to the escrow threshold being based on the token balance of the contract - a player will always be able circumvent sending the funds to the escrow by increasing the token balance of the contract.

Recommendation

Consider changing the mechanism for sending funds to the escrow to not rely on the liquidity in the contract. For example, having an absolute value instead.

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-4	Lack of a way to preview mint for LPs		
Asset	Bankroll.sol		
Status	Resolved		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description (POC)

LPs have no way to preview the minted amount before calling the `depositFunds()` function. They can only determine how many shares correspond to the deposited token amount after calling `depositFunds()`.

If the `shares` are worthless, the LP won't be able to foresee it. Consequently, they will lose their entire deposit.

Recommendation

Consider implementing a public view function, `previewMint()`, that takes in an amount of tokens as argument.

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-5	Lack of a way to preview redeem for LPs		
Asset	Bankroll.sol		
Status	Resolved		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description (POC)

If the value of the `shares` drops to 0 and an LP decides to withdraw, they will not receive any tokens in return, with no prior means of knowing this.
An LP holding `shares` has no incentive to withdraw when the `shares` have no value. Their interest lies in holding the `shares` until their value is at least greater than zero.

Consider the following scenario:
* Bob deposits 100 USDC to Bankroll and gets back 100e6 shares
* Admin debit to Alice 100 USDC
* Bob withdraws 100e6 shares
* However, Bob gets back 0 USDC because liquidity = 0

Recommendation

Implement a public view function, `previewRedeem()`, which accepts an amount of `shares` as an argument

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-6	Low `withdrawalDelay` and `withdrawalEventPeriod` enable front-running of `debit()`		
Asset	**Bankroll.sol: L51 L54**		
Status	**Resolved**		
Rating	**Severity: High**	**Impact: High**	**Likelihood: High**

Description

When the `debit()` function is called, `shares` lose value. An LP would have an incentive to withdraw before the `debit()` function is executed. Some parameters such as `withdrawalDelay` and `withdrawalEventPeriod` aim to mitigate this issue. However, there are no minimum requirements imposed for these values.

An LP could quite easily execute their call to the `withdrawStageOne()` function in one block and their call to the `withdrawStageTwo()` function in the next block, just before the `debit()` transaction

Recommendation

Consider setting a minimum value for `withdrawalDelay` and `withdrawalEventPeriod`

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

DBM-1	LP fees are sandiwchable		
Asset	DGBankrollManager.sol: L292		
Status	Resolved		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description (POC)

Stepwise jump in the value of the vault may allow an attacker to front-run `claimProfit()` to call `depositFunds()` and earn fees. The malicious actor can then claim these rewards in a relatively short period depending on `withdrawalWindowLength` and `withdrawalDelay`.
As a result, the attacker will receive equivalent rewards to those of an LP that deposited much earlier.

Recommendation

Consider implementing a “warmup period” where liquidity providers cannot accrue rewards or a depositFunds fee.

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

DGE-1	The `revertFunds()` function will revert when using USDT		
Asset	DGEscrow.sol: L169		
Status	Resolved		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description (POC)

In the `revertFunds()` function, the admin can revert funds to the bankroll in case of fraud. However, the following line will revert in case the token is USDT and the allowance is set for a non-zero value:

```
if (token.approve(entry.bankroll, escrowed[_id]))
```

NOTE: See the `approve()` function in the USDT contract.

Recommendation

Use `token.forceApprove()` instead of `token.approve()`. Consider also adding an else-statement so that a malicious player cannot wait for the event period to end and then call the `claimUnaddressed` function.

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

DGE-2		Risk of ID collision	
Asset		DGEscrow.sol: L93-102	
Status		Resolved	
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description (POC)

When `debit()` is called within the same block with the same player and operator, the escrow entry remains unchanged, resulting in identical IDs for two different calls.

```
// Create escrow entry
DGDataTypes.EscrowEntry memory entry = DGDataTypes.EscrowEntry(
    msg.sender,
    _operator,
    _player,
    _token,
    block.timestamp
);

// Encode entry into bytes to use for id of escrow
bytes memory id = abi.encode(entry);
```

As a result, `escrowed[id]` will be overwritten by the second `winnings`, thereby preventing the player from receiving the initial amount upon calling `releaseFunds()`.

Recommendation

Add a `nonce` to make sure that each `id` is unique.

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-7	Missing `gap` for contract upgrades		
Asset	Bankroll.sol		
Status	Acknowledged & closed		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The Bankroll.sol contract is currently missing a gap variable. Thus, during upgrades - this contract, risk having variables of their child contracts being overwritten.

Recommendation

Add a `_gap` variable to the contract.

Resolution

Acknowledged and closed

BKR-8	`ReentrancyGuardUpgradeble` is not initialised		
Asset	Bankroll.sol		
Status	Resolved		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

The ReentrancyGuardUpgradeble contract is never initialised in the Bankroll.sol contract. This means that the `nonReentrant` modifiers will not be initialised and thus not work.

Recommendation

Add the `__ReentrancyGuard.init()` to the `initialize()` function.

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-9		Player loses part of their debit when the ` _amount` is greater than `maxRisk`	
Asset		Bankroll.sol: L315	
Status		Acknowledged & closed	
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

When `debit()` is called, if the amount exceeds the maximum risk, the player will receive what remains in the bankroll balance, but the excess amount will be lost to them.
This encourages players to interact only with bankrolls with a high balance, thereby reducing the chances for those with a low balance to replenish their funds

Recommendation

If this behavior is unexpected, consider adding a mapping to keep track of the remaining prize amount that has not yet been sent to the player.

```
mapping(address => uint256) remainingPrize;
```

Add the remaining amount to the mapping in `debit()`:

```
if (_amount > maxRisk) {
      uint256 remainingAmount = _amount - maxRisk;
      _amount = maxRisk;
      remainingPrize(player) += remainingAmount;
      // Emit event that the bankroll is sweppt
      emit DGEvents.BankrollSwept(_player, _amount);
}
```

The admin can later call `debit()` to send the remaining amount to the player

Resolution

Acknowledged and closed.

BKR-10	Missing validation checks could DoS withdrawals		
Asset	Bankroll.sol: L315		
Status	Resolved		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

There are currently three time checks in place for withdrawals: `withdrawalDelay`, `withdrawalWindowLength`, `withdrawalEventPeriod`. The admin is the only role that can set them, however there are checks missing to make sure that the times are in certain time ranges relative to each other. For example, if `withdrawalDelay` is greater than `withdrawalWindowLength` then the LP wouldn't be able to withdraw their funds in the function `withdrawalStageTwo()`.
Furthermore, if `withdrawalWindowLength` is set to 0, `withdrawalStageTwo()` will revert every time due to the following check:

```
if( block.timestamp < withdrawalInfo.timestamp + withdrawalDelay || block.timestamp >
withdrawalInfo.timestamp + withdrawalWindowLength ) revert DGErrors.OUTSIDE_WITHDRAWAL_WINDOW();
```

Recommendation

Add the following checks **`setWithdrawalDelay()`**

```
if(_withdrawalDelay > withdrawalWindowLength) revert DGErrors.WITHDRAWAL_TIME_RANGE_NOT_ALLOWED
```

`setWithdrawalWindow()`

```
if(_withdrawalWindow < withdrawalDelay) revert DGErrors.WITHDRAWAL_TIME_RANGE_NOT_ALLOWED
if(_withdrawalWindow > withdrawalEventPeriod) revert DGErrors.WITHDRAWAL_TIME_RANGE_NOT_ALLOWED
```

`setWithdrawalEventPeriod()`

```
if(_withdrawalEventPeriod < withdrawalWindow) revert DGErrors.WITHDRAWAL_TIME_RANGE_NOT_ALLOWED
```

If withdrawals are not supposed to be restricted by setting `withdrawalWindowLength` to zero, prevent setting excessively low values for `withdrawalWindowLength` in `setWithdrawalWindow()`
Otherwise consider reverting `withdrawlStageOne()` if `withdrawalWindowLength` is set to zero:

```
if(withdrawalWindowLength == 0) revert DGErrors.WITHDRAWALS_NOT_ALLOWED;
```

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

DGF-1		Incorrect branch of OZ library used in upgradeable contract	
Asset		DGBankrollFactory.sol: L5	
Status		Resolved	
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

DGBankrollFactory.soll uses Clones 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 ClonesUpgradeable library instead.

```
import {ClonesUpgradeable} from "@openzeppelin/contracts-upgradeable/proxy/ClonesUpgradeable.sol";
```

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-11	Calling `debit()` on a bankroll with a `maxRisk` of 0 has no effect		
Asset	**Bankroll.sol: L312**		
Status	**Resolved**		
Rating	**Severity: Low**	**Impact: Low**	**Likelihood: Low**

Description

If a bankroll has a `maxRisk` set to zero and the `debit()` function is called, the `_amount` will be adjusted to 0. In this scenario, 0 tokens will be sent to the player, and the `GGR` will remain unchanged as it is reduced by zero. The same applies to the `ggrOf[_operator]`.

Recommendation

Consider reverting if `maxRisk` is 0.

```
if (maxRisk == 0) revert DGErrors.MAX_RISK_ZERO;
```

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

DBM-2	Excessively low `ggrOf` values will not generate any fees		
Asset	**DGBankrollManager.sol**		
Status	**Resolved**		
Rating	**Severity: Low**	**Impact: Medium**	**Likelihood: Low**

Description

In the `claimProfit()` function, the transaction will revert if the `GGR` is less than 1.

```
if (GGR < 1) revert DGErrors.NOTHING_TO_CLAIM();
```

However, if the fee is less than 10% and there is only one operator, there will be nothing to claim if the `GGR` is below 10 wei. Moreover, if there are multiple operators, each with a `ggrOf` less than 10 wei, the fees will be rounded down to 0 in the for loop each time. The total `ggrOf` will then be accounted for in the `amount` variable.

```
amount += uint256(bankroll.ggrOf(operators[i])) - ((lpFeeOf[_bankroll] *
uint256(bankroll.ggrOf(operators[i]))) / DENOMINATOR);
```

Recommendation

Consider reverting if the `GGR` is below 10.

Consider calculating the amount to transfer from the bankroll after the for loop

```
uint256 amountToTransfer = totalAmount - ((lpFeeOf[_bankroll] * totalAmount) / DENOMINATOR);
// transfer the GGR to DeGaming
token.safeTransferFrom(_bankroll, deGaming, amountToTransfer);
```

Resolution

This issue is resolved as of commit 16d90e33f41d61d0aaf4cd788a59a216ec4ab778

DGE-3	Players have no way of knowing the `eventPeriod` in `DGEscrow`		
Asset	**DGEscrow.sol:L32**		
Status	**Resolved**		
Rating	**Severity: Low**	**Impact: Low**	**Likelihood: Low**

Description

The visibility of `eventPeriod` is not specified. The default visibility level for state variables is internal. Since there is no getter function for this variable, players do not have access to it and cannot know when they have the opportunity to call the `claimUnaddressed()` function.

Recommendation

Consider setting the visibility of `eventPeriod` state variable to public or creating a `getEventPeriod()` view function

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

BKR-12	Not possible to debit Account Abstraction wallets		
Asset	Bankroll.sol: L308		
Status	Resolved		
Rating	Severity: Informational		

Description

Due to the below check it is currently not possible for smart contract wallets to be implemented under the ERC4337 standard. since those wallets have a code size.

```
if (_isContract(_player)) revert DGErrors.NOT_AN_EOA_WALLET();
```

Recommendation

Consider if the debit should strictly cater towards EOA

Resolution

This issue is resolved as of commit 3b4733e68d83584b93b49b73757e6bf4da0d2d2a

GLOBAL-1	Contract size check vulnerability		
Asset	-		
Status	Acknowledged & closed		
Rating	Severity: Informational		

Description

Checking code size on an address proves valuable when aiming to protect users, such as preventing them from sending tokens to contracts that might result in those assets being locked forever. However, when functions require the caller to be an EOA for security purposes, relying solely on this method is ill-advised.

During contract's `constructor` execution, `extcodesize` of the smart contract being deployed will return zero. No code exists at the contract address until the contract creation process concludes. Hence, as there is no code stored at contract addresses until `constructor` execution ends, any function called from a smart contract's `constructor` will bypass the target contract's `extcodesize` check.

If the objective is to restrict calls from other contracts, `(tx.origin == msg.sender)` could be utilized, but it also comes with its own limitations and potential vulnerabilities.
Fishing attacks will not be a risk if `tx.origin` is used solely to ensure that the `msg.sender` is not a contract

Recommendation

Do not rely on the `_isContract()` function in future upgrades if the goal is to prevent contracts from accessing a function.

Resolution

Acknowledged and closed.

GLOBAL-2	Following the ERC-4626 standard		
Asset	-		
Status	Acknowledged & closed		
Rating	Severity: Informational		

Description

The bankroll contracts utilize functionalities closely related to the workings of a vault such as depositing funds in exchange for shares. However, the contracts reviewed during this engagement have also shown that the vault is missing a few functionalities that might be beneficial to vaults.

For example, there currently is no way for a player to preview their share amount or their deposit amount. In addition, since the shares are currently represented as a mapping in the contracts, there's no way for the user to transfer shares.

For tokenized vaults, the current gold standard is ERC-4626 that enables the above functionalities among other benefits. Inheriting would potentially unlock new benefits in addition to lower the complexity of the current contracts.

Recommendation

Consider implementing the ERC-4626 standard.

Resolution

Acknowledged and closed.

Appendix

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