AUDITONE

Audit Report

13.12.2024

Table of Contents

CONTENTS

INTRODUCTION

Smart Contract Security Analysis Report

Note: This report may contain sensitive information on potential vulnerabilities and exploitation methods. This must be referred internally and should be only made available to the public after issues are resolved (to be confirmed prior by the client and AuditOne).

INTRODUCTION

X0f-unit, Senya123 and Fyvgsk, who are auditors at AuditOne, successfully audited the smart contracts (as indicated below) of Dark Mythos. The audit has been performed using manual analysis. This report presents all the findings regarding the audit performed on the customer's smart contracts. The report outlines how potential security risks are evaluated. Recommendations on quality assurance and security standards are provided in the report.

01-PROJECT DESCRIPTION

Dark Mythos is a fantasy trading card game (TCG) that integrates blockchain technology to offer gaming experience through the use of NFTs. The platform is designed to appeal to both trading card enthusiasts and fans of fantasy literature, incorporating rich narrative elements within its gameplay. Players in the Dark Mythos universe collect and trade digital cards, each embedded with exclusive stories that provide deeper insights into the game's lore and character backgrounds.

Dark Mythos is developed on immersive, handcrafted stories written by renowned fantasy author Marco Dülk, known for his detailed world-building and engaging narratives. The integration of these stories with the NFT cards allows players to unlock new chapters and story arcs as their collections grow, offering a dynamic and evolving gameplay experience.

The audit focuses on the new Arcane contracts.

02-Project and Audit Information

Term	Description
Auditor	X0f-unit, Senya123 and Fyvgsk
Reviewed by	Luis Buendia and Gracious Igwe
Type	NFT Gaming
Language	Solidity
Ecosystem	IOTA EVM compatible
Methods	Manual Review
Repository	https://github.com/DarkMythosWeb3/Dark-Mythos
Commit hash (at audit start)	995a1125b45a3253e68526a7f1a30cb1a250865d
Commit hash (after resolution)	41629a910e3da3a2429926df761c9fe6974a944c
Documentation	https://docs.dark-mythos.com/dark-mythos-gamepaper/general/introduction
Unit Testing	N/A
Website	https://dark-mythos.com/
Submission date	03/12/2024
Finishing date	13/12/2024

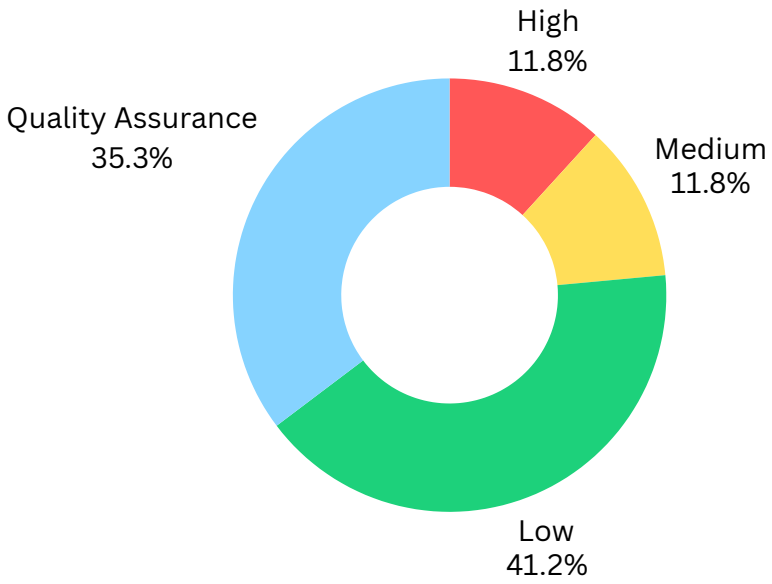
03-Contracts in Scope

Contract in scope:

- ArcaneBinding.sol
- ArcanseEssence.sol
- ArcaneVortex.sol
- Arcanum.sol
- ArcanumSale.sol
- ERC20DarkMythosEcoBoundArthas.sol
- PriceCalculator.sol

04-Executive summary

Dark Mythos smart contracts were audited between 04-11-2024 and 28-11-2024 by X0f-unit, Senya123 and Fyvgsk. Manual analysis was carried out on the code base provided by the client. The following findings were reported to the client. For more details, refer to the findings section of the report.



Issue Category	Issues Found	Resolved	Acknowledged
High	2	2	0
Medium	2	2	0
Low	7	6	1
Quality Assurance	6	6	0

05-Severity Definitions

Risk factor matrix	Low	Medium	High
Occasional	L	M	H
Probable	L	M	H
Frequent	M	H	H

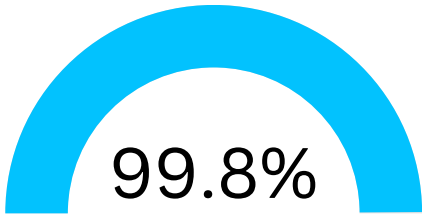
High: Funds or control of the contracts might be compromised directly. Data could be manipulated. We recommend fixing high issues with priority as they can lead to severe losses.

Medium: The impact of medium issues is less critical than high, but still probable with considerable damage. The protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions.

Low: Low issues impose a small risk on the project. Although the impact is not estimated to be significant, we recommend fixing them on a long-term horizon. Assets are not at risk: state handling, function incorrect as to spec, issues with comments.

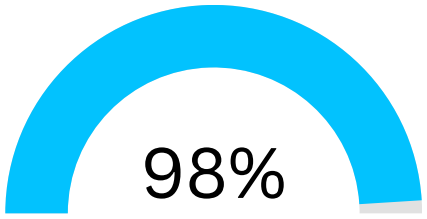
Quality Assurance: Informational and Optimization - Depending on the chain, performance issues can lead to slower execution or higher gas fees. For example, code style, clarity, syntax, versioning, off-chain monitoring (events etc.)

06-Audit Overview



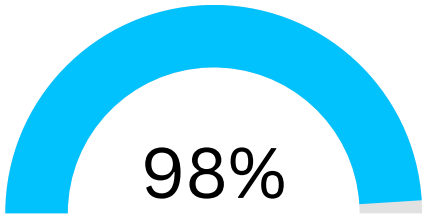
Security score

Security score is a numerical value generated based on the vulnerabilities in smart contracts. The score indicates the contract's security level and a higher score implies a lower risk of vulnerability.



Code quality

Code quality refers to adherence to standard practices, guidelines, and conventions when writing computer code. A high-quality codebase is easy to understand, maintain, and extend, while a low-quality codebase is hard to read and modify.



Documentation quality

Documentation quality refers to the accuracy, completeness, and clarity of the documentation accompanying the code. High-quality documentation helps auditors to understand business logic in code well, while low-quality documentation can lead to confusion and mistakes.

07-Findings

Finding: #1

Issue: Arcane balance tracking in ArcaneVortex contract can be tampered

Severity: High

Where: ArcaneVortex.sol#L85C1-L101C6

Impact: Each direct Arcanum transfer to the ArcaneVortex contract will irremediably create a gap between the real totalTokensProvided value and the real one.

Additionally, it will also break the providedTokensInCirculation() view function in two different ways:

- In the best case scenario, the providedTokensInCirculation() will return a value lower than the real one, since the totalTokensProvided - tokensInVortex() operation does not take into account the directly transferred tokens.
- If the Arcanum balance of ArcaneVortex contract is higher than the value of totalTokensProvided, the providedTokensInCirculation() function will remain unusable due to an integer underflow, hence reverting on every call.

It should be also considered that the malfunctioning of these view functions might directly affect the in-game metrics, potentially impacting the entire protocol.

Description: The ArcaneVortex contract is designed to manage Arcane transfers and track the balance of tokens provided and withdrawn. In order to achieve this, each token transfer performed with transferAndCall() or transferFromAndCall() functions will use the onTransferReceived() hook to update the value of totalTokensProvided:

https://github.com/DarkMythosIOTA/Dark-Mythos/blob/995a1125b45a3253e68526a7f1a30cb1a250865d/ERC20/contracts/ArcaneVortex.sol#L85C1-L101C6

Additionally, in this hook, a capability check is performed to ensure that only Coordinators with the **CanProvideTokens** capability are able to provide tokens:

https://github.com/DarkMythosIOTA/Dark-Mythos/blob/995a1125b45a3253e68526a7f1a30cb1a250865d/ERC20/contracts/ArcaneVortex.sol#L95

However, this capability check can be trivially bypassed simply by using **transfer()** or **transferFrom()** methods. In addition, the provided tokens tracking will be tampered forever.

It must be noted that Arcanum tokens are EcoBound and not Releasable. Only operators with the **CanTransferEcoBoundToken** capability would be able to freely transfer Arcanum which might decrease the likelihood of this finding since Coordinators should be trusted roles, if it were not for the normal operations of contracts like **ArcaneBinding**, which will perform regular **transfer()** to the **penaltyReceiver** address, which is contemplated to be the **ArcaneVortex** address, effectively maintaining the likelihood of this finding:

https://github.com/DarkMythosIOTA/Dark-Mythos/blob/995a1125b45a3253e68526a7f1a30cb1a250865d/ERC20/contracts/ArcaneBinding.sol#L195

Recommendations: In order to completely fix this issue, and to ensure a correct Arcanum tracking on the **ArcaneVortex** contract, it should be mandatory to use **transferAndCall()** or **transferFromAndCall()** when sending tokens to the **ArcaneVortex** contract.

This could be achieved by adding a check on the **transfer()** and **transferFrom()** functions from **ERC20DarkMythosEcoBoundArthas** contract, reverting if the destination address is the **ArcaneVortex** contract address.

Another alternative partial fix would be to enforce the use of **transferAndCall()** and **transferFromAndCall()** along every contract in the protocol, although this vulnerability could still be triggered by authorized Coordinators.

Status: Resolved.

Finding: #2

Issue: Usage of transfer() to transfer native ETH

Severity: High

Where: ArcanumSale.sol#L301

Impact: Since send() and transfer() functions implement a hard gas limit of 2300, the ETH transfer may revert in the following scenarios:

- The destination is a smart contract that either does not have a payable function, or it has a payable function that requires more than 2300 gas units to execute.
- The destination is a smart contract that either lacks a payable fallback function, or it has such a function that also consumes more than 2300 gas units.
- The destination is a smart contract accessed through an intermediate proxy contract, which raises the gas requirements beyond 2300 units. Additionally, the complexity of the destination can increase in cases like a multisig wallet that operates using more than 2300 gas units.
- Changes or forks in Ethereum may lead to increased gas fees beyond what the .transfer() function accommodates. This method imposes a strict reliance on the appropriateness of 2300 gas units now and in the future.

The risk impact of this finding has been raised to High due to the confirmation of fundsRecipient address to be a multisig wallet, and due to the inability to perform any Arcanum sale, which is the main purpose of the affected contract and a key feature to the protocol's overall functioning.

Description: The **ArcanumSale** contract is using **transfer()** instead of **call()** for transferring native ETH both to the **fundsRecipient** address (which is confirmed to be a multi-sig).

Recommendations: Using **call()** instead of **transfer()** is recommended.

Status: Resolved.

Finding: #3

Issue: Excess Ether received gets locked in the contract

Severity: Medium

Where: PriceCalculator.sol#L89

Impact: Since no refund mechanism is implemented, the excess of **ETH** sent in **msg.value** will remain locked in the contract.

Description: When any user calls **PriceCalculator.updatePriceFeeds()**, a **msg.value** must be passed in order to pay the price update fee from Pyth oracle.

Controls to ensure that **msg.value** is equal or greater than **updateFee** are in place, but there is no mechanism to reimburse caller with the excess of **ETH** sent in **msg.value**.

Recommendations: It is recommended to either check **msg.value == updateFee** or to implement an excess refund mechanism on both functions.

Status: Resolved.

Finding: #4

Issue: If user unbind the token he could accidentally loose all his rewards

Severity: Medium

Where: ArcaneBinding.sol#L172-L209

Impact: Rewards is being lost.

Description: User could **_claimEssences** when he calls the **claimEssences** or **addTokensToBinding**, which would correctly update his reward balance. However, the **_claimEssences** check isn't implemented in the **unbindTokens** function and if the user has some pending rewards, during the unbinding of the tokens, the user binding is deleted, thus all the rewards also. It is incorrect behaviour.

For example, we can take a look at Curve finance **withdraw** function, which serves the same purpose as **unbindTokens**. However, it correctly implements the checkpoint function, which save the global state and potentially account for the newly distributed rewards.

https://github.com/curvefi/curve-dao-contracts/blob/567927551903f71ce5a73049e077be87111963cc/contracts/VotingEscrow.vy#L488

Recommendations: Ensure that during the token unbind the **_claimEssences** is executed, so no rewards are being lost.

Status: Resolved.

Finding: #5

Issue: Pyth confidence intervals are ignored

Severity: Low

Where: PriceCalculator.sol#L105-L120

Impact: Incorrect price propagation.

Description: Pyth price data includes a confidence interval (**σ**) that indicates the uncertainty around the reported price (**p**), similar to the standard deviation of a price's probability distribution. The Pyth documentation suggests using this interval for improved security. For instance, the protocol can calculate **σ / p** to measure price uncertainty and restrict user actions if this ratio exceeds a defined threshold.

Currently, the protocol does not account for this confidence interval.

Recommendations:

```
+       if (priceData.conf > 0 && (priceData.price / int64(priceData.conf) < minConfidenceRatio)) {
+           revert LowConfidencePyth();
+       }
```

Status: Resolved.

Finding: #6

Issue: Consecutive claims bonus essences are always awarded

Severity: Low

Where: ArcaneBinding.sol#L433

Impact: Since the number of days to be consecutive should be at least **2**, granting consecutive days bonus since day 1 could lead to unexpected token inflation or affect the intended in-game economy.

Description: The **ArcaneBinding** contract allows users to bind their Arcane tokens for a certain duration in order to being able to claim Essences. The amount of claimable essences will vary depending on the initial binding amount and on the bind duration.

In addition, a bonus boost to the claimable essences is applied if the player claims their essences on consecutive days.

However, and due to how the **newConsecutiveClaimDays** value is calculated in **_calculatePendingEssences()** , the users will obtain a consecutive claim bonus every time. The stacked bonus will be lost if no consecutive claims were performed, but the value of **newConsecutiveClaimDays** will always be at least **1**.

Recommendations: It is recommended to adjust the consecutive days calculation logic to adjust to the intended functionality.

Status: Resolved

Finding: #7

Issue: Zero duration rates can be set but will never be effectively used

Severity: Low

Where: ArcaneBinding.sol#L106

Impact: This finding will prevent the **0** duration rates from generating any claimable Essence.

Description: The **setDurationRate()** function allows the owner to set a certain essence generation given the duration of the Arcane binding. This function allows to set a valid rate for a **0** duration binding, which could be used for certain special in-game events.

However, due to the implemented logic in **_calculatePendingEssences()** function (ArcaneBinding.sol#L420C1-L423C94) , the calculated **currentDay** will always be equal to **lastClaimDay**, causing this function to always return **0** as **essencesToAward**.

Recommendations: If **0** duration rates will be used, it is recommended to adjust the logic of **_calculatePendingEssences()** for this edge case. On the other hand, if **0** will never be set as a valid value, it is recommended to add a zero value check on **setDurationRate()** in order to prevent the owner from setting unusable rates.

Status: Resolved.

Finding: #8

Issue: Multiple centralization risks

Severity: Low

Where: Scoped contracts

Impact: This centralization will result in a complete protocol take over if the owner's account is compromised or if the owner acts maliciously.

Description: It has been noted that almost every key role and the ability to control the key functionalities for the protocol is centralized on the contract's **owner**, which poses a significant security risk.

A single owner would be capable of performing at least the following actions:

- Set the rate for a specific binding duration (**setDurationRate()**).
- Update external addresses for tokens and contracts (**setExternalAddresses()**).
- Update the penalty percentage for early token unbinding (**setPenaltyPercentage**()).
- Pause and unpause the contract, affecting all user interactions (**pause(), unpause()**).
- Update coordinator capabilities, allowing them to mint or burn essences, provide or withdraw tokens, or release or transfer eco-bound tokens (**updateCoordinatorCapability()**).
- Transfer ownership of the contracts (**transferOwnership()**).
- Update the address of the Arcanum token (**setArcanumAddress()**).
- Set or update special offers for token sales (**setSpecialOffer**()).
- Set the regular token sale price (**setRegularTokenPrice()**).
- Mint new tokens to a specified address (**mint()**).

- Manage accepted payment tokens, including setting their price feed IDs and acceptance status (**manageAcceptedPaymentToken()**).
- Update the Pyth contract address used for price feeds (**setPythContract()**).
- Update the maximum age of price data (**setMaxPriceAge()**).

Recommendations: Implementing decentralized governance or multi-signature requirements for critical functions can help mitigate these risks. Additionally, transparency in owner actions and regular audits can enhance trust.

Status: Resolved.

Finding: #9

Issue: setMaxConsecutiveClaimDays can be abused by the owner

Severity: Low

Where: ArcaneBinding.sol#L133-L136

Impact: User left with no bonuses.

Description: In the **setPenaltyPercentage** we can see that the check related to the fair/reasonable penalty exists. It gives the contract more trust and increase the confidence from the user's side.

```
if (newPenaltyPercentage > 25) revert InvalidPenaltyPercentage();
```

However, in the **setMaxConsecutiveClaimDays** there is no similar checks. What could happened, is that the owner could abuse the users by setting the **maxConsecutiveClaimDays** to **0**, which would prevent from bonus distribution.

Recommendations: Ensure that the **maxConsecutiveClaimDays ≠ 0.**

Status: Resolved.

Finding: #10

Issue: safeTransfer should be used in the Vortex + other instances.

Severity: Low

Where: ArcaneBinding.sol#L194-L195
ArcaneVortex.sol#L80

Description: The current solidity standards require to utilize the safeTransfer library, to avoid the unintended consequences. In the **ArcaneBinding**, **ArcaneVortex**, the transfer of the arcanum token is implemented without safe transfer library which is discouraged.

Recommendations: Implement **safeTransfer** library on the arcanum token, as well as on whitelisted ones.

Status: Resolved.

Finding: #11

Issue: Fees in **ArcanumSale.purchaseTokens** can be bypassed

Severity: Low

Where: ArcanumSale.sol

Impact: Users may pay less fee than expected.

Description: The **purchaseTokens()** function allows users to get some arcanum using a valid token, the amount they want to spend, if its a special offer and the **priceUpdateData**. This parameter is used to update the price feeds and obtain the fee for which will be subtracted from the total amount to spend. It is possible to use a **priceUpdateData** from other token with lower fees to maximize the obtained tokens, as it will not validate the token used to pay with the token from **priceUpdateData**.

Recommendations: It is recommended to check that the used token is the same of **priceUpdateData**. For this, it is possible to check that the **priceFeedId** stored in **acceptedPaymentTokens** is the same that the one used in **priceUpdateData.**

Status: Acknowledged.

Finding: #12

Issue: Low granularity when setting **penaltyPercentage**.

Severity: Quality Assurance

Where: ArcaneBinding.sol#L186

Impact: Not having enough granularity might impact the owner's ability to fine-tune the functioning of the protocol.

Description: It has been noted that, when setting **penaltyPercentage** value, a granularity of **1%** is used. However, other parameters such as **ratings** or **bonus multiplier** are using a granularity of **0.01%**.

Recommendations: It is suggested to use the same granularity criteria for every parameter, in order to both allow more flexibility when setting its values and to prevent potential misusage.

Status: Resolved.

Finding: #13

Issue: Unnecessary checks

Severity: Quality Assurance

Where: ArcaneBinding.sol#L189

Impact: Having unnecessary statements both increase gas usage and reduce code readability.

Description: The following line was found on the **unbindTokens()** function from the **ArcaneBinding** contract. The purpose of this line is to prematurely revert if **ArcaneBinding** does not have enough funds in order to fulfill a token unbinding.

Two things must be noted on this statement:

1. Instead of **returnedAmount + penaltyAmount**, **amount** could be used instead.
2. This check is unnecessary since if there is not enough balance, the transaction will revert anyways.

Recommendations: It is recommended to optimize the code in order to reduce gas usage and increase code readability and to check for additional instances of this finding.

Status: Resolved.

Finding: #14

Issue: Lack of precision in Bonus Essences calculation.

Severity: Quality Assurance

Where: ArcaneBinding.sol#L433

Impact: The inaccuracy when calculating **bonusEssences** or directly its rounding down to zero might affect both player's economy and the functioning of the overall game.

Description: An edge case when calculating **bonusEssences** within the **_calculatePendingEssences()** function was detected. However, this finding was marked as QA due to its extremely low likelihood.

When calculating **bonusEssences**, the following code is used: **bonusEssences = (newConsecutiveClaimDays * bonusMultiplier * essencesToAward) / RATE_PRECISION;**

Since the **essencesToAward** value is already normalized (divided by **RATE_PRECISION**), there is a risk of rounding down to zero the value of **bonusEssences** if low values of **essencesToAward** are used. Specifically, this edge case would be triggered if **(newConsecutiveClaimDays * bonusMultiplier * essencesToAward) < RATE_PRECISION.**

Please note that, if **RATE_PRECISION** is still smaller than the other factor but it still has a closer order of magnitude, the **essencesToAward** calculation will be inaccurate.

Recommendations: If this scenario is plausible, adding additional precision decimals (or using the not-normalized **essencesToAwardAmount**) to perform the calculations is recommended.

Status: Resolved.

Finding: #15

Issue: Using enum with only a single value

Severity: Quality Assurance

Where: ArcanumSale.sol#L15

Impact: Using an enum with only a single value increases gas usage and reduces code simplicity and readability.

Description: It has been detected that the **Capability** enum from **ArcanumSale** contract was declared with only a single value.

Recommendations: Although this enum was probably imported from the **ERC20DarkMythosEcoBoundArthas** contract, it could be substituted by a single bool variable.

Status: Resolved.

Finding: #16

Issue: Variables could be set as **Immutable.**

Severity: Quality Assurance

Where: ERC20DarkMythosEcoBoundArthas.sol#L24

Impact: Decreased code readability and increased gas usage.

Description: Some variables defined as regular state variables could be declared as **Immutable** in order to save gas and improve code readability.

Recommendations: These variables could be declared as **Immutable** in order to save gas.

Status: Resolved.

Finding: #17

Issue: There is no WETH,WBTC,WIOTA pyth feed on the IOTA chain

Severity: Quality Assurance

Where: PriceCalculator

Impact: Protocol can't be integrated with the tokens that expected to be supported.

Description: On the IOTA chain, some of the tokens that are expected to be whitelisted, simply don't exists. It would be problematic to implement such tokens in the protocol, since it is not clear when these feeds will be officialy listed on IOTA chain. You can check the avaliability of the tokens feed here:

https://api-reference.pyth.network/price-feeds/evm/getEmaPriceUnsafe

Recommendations: Take into account that some of the tokens could not be available on the IOTA chain.

Status: Resolved.

08 - Disclaimer

The smart contracts provided to AuditOne have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). The ethical nature of the project is not guaranteed by a technical audit of the smart contract. Any owner-controlled functions should be carried out by the responsible owner. Before participating in the project, all investors/users are recommended to conduct due research.

The focus of our assessment was limited to the code parts associated with the items defined in the scope. We draw attention to the fact that due to inherent limitations in any software development process and product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which cannot be free from any errors or failures. These preconditions can impact the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure, which adds further inherent risks as we rely on correctly executing the included third-party technology stack itself. Report readers should also consider that over the life cycle of any software product, changes to the product itself or the environment in which it is operated can have an impact leading to operational behaviors other than initially determined in the business specification.

Contact

auditone.io

@auditone_team

hello@auditone.io

A trust layer of our
multi-stakeholder world.