HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Authic Labs
Date: 02 Aug, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for Authic Labs
Approved By	Arda Usman | Lead Solidity SC Auditor at Hacken OU
Tags	ERC1155 token; NFT marketplace;
Platform	EVM
Language	Solidity
Methodology	[Link]()
Website	https://authic.io/
Changelog	15.06.2023 - Initial Review 02.08.2023 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Authic Labs (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

AuticLabs - is an NFT market place with following smart contracts:

- *FactoryV2* - is a factory smart contract for creating and managing ERC-1155 token collections. It is implemented with several features to enhance its security and functionality. The factory contract allows the creation of new token collections, each assigned a minter role to a designated marketplace.
- *PrimaryMarketPlaceV2* - is a marketplace smart contract that allows the purchase of ERC-1155 tokens at a fixed price. It includes the functionality to withdraw commission and pause, unpause the contract.
- *AuthicERC1155* - an implementation of a smart contract for ERC1155, which is a token standard on the Ethereum blockchain. The contract allows for the minting, transfer, and burning of tokens. It also includes royalty functionality, meaning that certain addresses can receive a defined percentage of transactions involving specific tokens. Additionally, it includes a mechanism to check if an address is permitted to mint tokens using a Merkle proof.
- *IERC1155* - an interface for ERC1155 smart contract.
- *IERC1155Receiver* - an interface of ERC1155 token receiver.
- *IFactory* - an interface for FactoryV2 smart contract.
- *IPrimaryMarketplace* - an interface for PrimaryMarketPlaceV2 smart contract.

Privileged roles

FactoryV2:

- Admin Role:
  - The role is granted to the contract deployer initially and can perform critical operations such as: Pausing and unpausing the contract.
  - Modifying the marketplace and validator addresses.

PrimaryMarketPlaceV2:

- Admin Role:
  - The role is granted to the contract deployer initially and can perform critical operations such as: withdraw fees, set validator pause and unpause smart contract.

AuthicERC1155:

- Admin Role:

- The role is granted to the contract creator during the contract deployment. It allows the holder to perform administrative functions such as: updating the Merkle root for whitelisting and setting the maximum supply for a token. It can add new token types to the contract.
- The role is granted to the marketplace address during the contract deployment. This allows the holder to mint new tokens to a specified address.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Technical description is inadequate:
  - Run instructions are provided.
  - Technical specification is provided.
  - NatSpec is sufficient.
- Functional requirements are present, but only in a limited capacity:
  - Overall system requirements are provided.
  - No roles description.
  - Use cases are not fully described.

Code quality

The total Code Quality score is **9** out of **10**.

- Best practice violation.

Test coverage

Code coverage of the project is **83.71%** (branch coverage), with a mutation score of 77%.

- Deployment and basic user interactions are covered with tests.
- Interactions by several users are tested thoroughly.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the "Findings" section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.9**.The system users should acknowledge all the risks summed up in the risks section of the report.

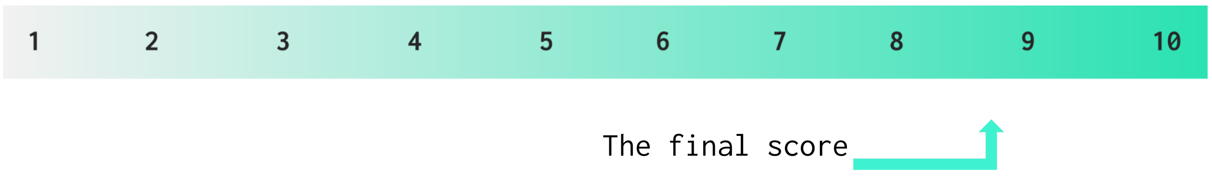

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
15 June 2023	2	5	2	2
02 August 2023	0	0	0	0

Risks

- Iterating over a dynamic array populated with custom tokenId can lead to **Gas limit denial of service if the number of tokenIds gets too big.**
- Payment **Splitter.sol** is deprecated by OpenZeppelin, this can cause problems in the later stages of the project's lifecycle.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Passed	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Not Relevant	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Failed	

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Failed	I09
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

■■■■ Critical

C01. Compilation Error

Impact	High
Likelihood	High

The function is available for calling by everyone. Field *admin* is The provided code contains compilation errors: "Undeclared identifier. "addNewToken" is not (or not yet) visible at this point."

Path: ./contracts/ERC1155/CustomERC1155.sol : addNewToken();

Recommendation: Mark the function as public.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

C02. Requirements Violation

Impact	High
Likelihood	High

The fixed-price purchase feature distributes the funds deposited for purchase by the user to a pre-determined list of addresses, called payees. This creates some uncertainty in the code, if payees contain seller and commission receivers then, the royalties are not distributed. If they are royalty recipients, then the seller is not getting paid. In addition to this, if payees contain royalty recipients, the amount is most probably different from the actual royalties, since none of the royalty calculating functions are being called. This can lead to payment imbalances.

Path: ./contracts/PrimaryMarketplace.sol : purchaseFixedPrice

Recommendation: Re-implement purchaseFixedPrice function logic.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

■■■ High

H01. Funds Lock - Leftover Funds

Impact	High
Likelihood	Medium

The _validateTransaction function validates the incoming minting request, while doing so, it checks if the deposit funds are enough with the following require statement "require(msg.value >= message.payment + message.commission, "Not enough payment");". As seen from this, if the caller sends more than required, the left-over amount is not returned to the caller. The contract accepts token deposits but lacks a withdrawal mechanism, which can result in funds being locked in the contract.

Path: ./contracts/ERC1155/CustomERC1155.sol : addNewToken()

Recommendation: Return the left-over amount to the caller or make the required funds exact.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

H02. Requirements Violation

Impact	High
Likelihood	Medium

The mintTo() function contains a comment for royalty setting, but the royalties are not being set there. The current implementation requires royalties to be set manually by the default admin role. This can lead to some of the royalties to be not set. The implementation of the system or function does not adhere to the high-level, broad system, technical, or functional requirements. This means that the implementation may not meet the intended purpose or design of the system, which could result in major issues down the line.

Path: ./contracts/PrimaryMarketplace.sol : mintTo()

Recommendation: Implement royalties in the mintTo() function.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

Medium

M01. Missing Events

Impact	Medium
Likelihood	Medium

Functions that produce a crucial impact on the state of the smart contract, should emit events.

Paths: ./contracts/ERC1155/Factory.sol : setMarketplace(), setValidator(), initialize(); ./contracts/ERC1155/PrimaryMarketplace.sol : _transferPayment(), setValidator(), withdrawCommission(), initialize(); ./contracts/ERC1155/CustomERC1155.sol : setApprovalForAll(), addNewToken(), burn();

Recommendation: Add missed events.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

M02. Unchecked Return Value

Impact	Medium
Likelihood	Medium

Return value of the function is not being validated. In the absence of proper validation, a situation might arise where function returns an error, yet the transaction still proceeds to completion.

Path: ./contracts/PrimaryMarketplace.sol : _transferPayment();

Recommendation: Incorporate a check for the return value of the function to ensure it executes correctly.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

M03. Best Practice Violation: Usage of built-in Transfer

Impact	Medium
Likelihood	Medium

The built-in transfer and send functions process a hard-coded amount of Gas. In case the receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

Path: ./contracts/PrimaryMarketplace.sol : _transferPayment();

Recommendation: Replace transfer and send functions with call or provide special mechanism for interacting with a smart contract.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

M04. Inefficient Gas Modeling

Impact	Medium
Likelihood	Medium

The addNewToken() function check if the parameter _tokenId is equal to uint type max. After it tokenId value is calculated with the data stored in the smart contract. If remove this redundant function argument and if statement the business logic of this function left the same, but transaction Gas cost would be decreased.

Path: ./contracts/ERC1155/CustomERC1155.sol : addNewToken()

Recommendation: Remove redundant functionality

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

M05. Highly Permissive Role Access

Impact	Medium
Likelihood	Medium

The default admin role can set royalty info (receiver and bps) for a tokenId anytime without notifying anyone If a key leak were to occur, the potential consequences could be significant, potentially leading to security breaches and undermining the overall integrity of the system.

Path: ./contracts/ERC1155/CustomERC1155.sol : setRoyaltyInfoForToken()

Recommendation: To ensure transparency and accountability, it is advised to provide a comprehensive explanation of highly-permissive access in the system's public documentation. This would help to ensure that users are fully informed of the implications of such access and can make informed decisions accordingly.

Found in: e67250b

Status: Mitigated (Revised commit: 2a1e2cf. With Customer notice: *OPERATOR can set token level royalty after minting a token for secondary marketplace sales by calling setRoyaltyInfoFor function in AuthicERC1155 contract. This poses a potential risk where if the OPERATOR or DEFAULT_ADMIN account is compromised then royalty percentages can be set from the compromised account thus disrupting the normal functions for collection secondary sales royalties.*)

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Low

Address parameters are used without checking against the possibility of 0x0. This can lead to unwanted external calls to 0x0.

Paths: ./contracts/ERC1155/CustomERC1155.sol : constructor(), setApprovalForAll(), isApprovedForAll(), mintTo(), setDefaultRoyaltyInfo(), setRoyaltyIfoForToken(), setRoyaltyInfoForToken(); ./contracts/ERC1155/Factory.sol : initialize(), setMarketplace(), setValidator(); ./contracts/ERC1155/PrimaryMarketplace.sol : initialize(), setValidator();

Recommendation: Implement zero address checks.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

L02. Function Naming Mismatching

Impact	Low
Likelihood	Low

The mintCollection() function contradicts actual code behavior. According to the implementation, the function should register a collection; however, according to the name, it should have minted it.

This can cause users to have wrong assumptions.

Path: ./contracts/Factory.sol : mintCollection()

Recommendation: Rename the function.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

Informational

I01. Floating Pragma

The project uses floating pragmas ^0.8.0. This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths: ./contracts/ERC1155/Factory.sol
./contracts/ERC1155/PrimaryMarketplace.sol
./contracts/ERC1155/CustomERC1155.sol
./contracts/ERC1155/interface/IERC1155.sol
./contracts/ERC1155/interface/IERC1155Receiver.sol
./contracts/ERC1155/interface/IFactory.sol
./contracts/ERC1155/interface/IPrimaryMarketplace.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I02. Style Guide Violation: Order of Functions

The provided projects should follow the official guidelines. There is an order of function violation.

Paths: ./contracts/ERC1155/CustomERC1155.sol : ./contracts/ERC1155/Factory.sol : ./contracts/ERC1155/PrimaryMarketplace.sol :

Recommendation: Follow the official Solidity guidelines.

Found in: e67250b

Status: Reported (Revised commit: 2a1e2cf. ./contracts/ERC1155/AuthicERC1155 : public *setRoyaltyInfoForToken()* is above external *getDefaultRoyaltyInfo()*)

I03. Redundant Virtual Keyword

The functions are declared virtual, but since this contract is not meant to be inherited, it can be removed.

Path: ./contracts/ERC1155/CustomERC1155.sol : supportsInterface(), uri(), burn(), burnBatch(), _burn(), _burnBatch(), royaltyInfo();

Recommendation: Remove the redundant virtual keyword.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I04. Functions That Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Paths: ./contracts/ERC1155/CustomERC1155.sol : updateRoot(), supportsInterface(), balanceOfBatch(), setApprovalForAll(), safeTransferFrom(), safeBatchTransferFrom(), burn(), burnBatch(), setDefaultRoyaltyInfo(); ./contracts/ERC1155/Factory.sol : initialize(), validator(), marketplace(); ./contracts/ERC1155/PrimaryMarketplace.sol : initialize(), validator();

Recommendation: Use the external attribute for functions never called from the contract.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I05. Style Guide Violation: Naming Mismatch

The names of the contracts should be equal to the file names.

Paths: ./contracts/ERC1155/PrimaryMarketplace.sol ./contracts/ERC1155/Factory.sol ./contracts/ERC1155/CustomERC1155.sol

Recommendation: Change filenames according to smart contract naming.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I06. Unused Variable

Some state variables are declared in the smart contract code, but never used.

Paths: ./contracts/ERC1155/PrimaryMarketplace.sol : _owner ./contracts/ERC1155/CustomERC1155.sol : owner, MAX_INT

Recommendation: Remove unused variables.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I07. Unused Function

The function _transferPayment() is presented in the code, but never used.

Paths: ./contracts/ERC1155/PrimaryMarketplace.sol : _transferPayment();

Recommendation: Remove unused function from the code.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I08. Non-finalized Code

Code should not contain development comments. This means that the code is not finished and is not in the final stage of development.

Paths: ./contracts/ERC1155/PrimaryMarketplace.sol

Recommendation: Remove development comments from the code. Finalize code.

Found in: e67250b

Status: Fixed (Revised commit: 2a1e2cf)

I09. Boolean Equality

Boolean variables in a smart contract are unnecessarily compared to true or false. This practice can lead to inefficient code and excessive gas usage.

Paths: ./contracts/ERC1155/AuthicERC1155.sol : mintTo();

Recommendation: Use Boolean variables directly instead of comparing them to true or false, as this can save gas and make the contract more efficient.

Found in: 2a1e2cf

Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://gitlab.com/a7717/authic-smart-contracts
Commit	e67250b
Whitepaper	-
Requirements	-
Technical Requirements	-
Contracts	File: contracts/ERC1155/Factory.sol SHA3: 529c18b04693c81981d5d8ae7b70c647e1cf59a3016c940a44576a5da42f6bf1 File: contracts/ERC1155/PrimaryMarketplace.sol SHA3: 1e44756764f047068af795ec167a476281117fc7f93aa74f4a88c674929046df File: contracts/ERC1155/CustomERC1155.sol SHA3: c020986e7ec07c7b54096881e0a9ec9e388018dfa1f58c47c88c924cea62f3a3 File: contracts/ERC1155/interface/IERC1155.sol SHA3: 1bc9e4d7ccf31ead5a835c542f504abbe37e810bcb92b3ed1defa29ceeff0667 File: contracts/ERC1155/interface/IERC1155Receiver.sol SHA3: 358044e24169213fa315af0c618d5e683b7bff6d8bb7ed5d609a23fc62a1061b File: contracts/ERC1155/interface/IFactory.sol SHA3: 27530ddcacff7b995456d048636b3d19ab17d9c59fab21d411e8dbd670f7cc08 File: contracts/ERC1155/interface/IPrimaryMarketplace.sol SHA3: f7b59d0ea4e11d5b519ee6da26c30a3f74bf20acd7732f21585c37dd854b4964

Second review scope

Repository	https://github.com/Authic-Labs/authic-smart-contracts
Commit	2a1e2cf
Whitepaper	-
Requirements	https://drive.google.com/file/d/1CIi4caKRRUWM3d4qAUWK26ja2jn0NaRp/view?usp=sharing
Technical Requirements	https://github.com/hknio/authic-smart-contracts/blob/audit/review-1/README.md
Contracts	File: contracts/ERC1155/AuthicERC1155.sol SHA3: 638822d61f93551cf8067e0f7165c7ed5149bdacd6ed746c6977d682 File: contracts/ERC1155/FactoryV2.sol SHA3: db2cbe667d7abdb2818cd28d32318f5d5b90b4fccdce99c99b18ff6b

	File: contracts/ERC1155/PrimaryMarketplaceV2.sol SHA3: 512c48070f321dc2b4dcb661218caeb07d39abfdb18205ba31d4ef1e File: contracts/ERC1155/interface/IPrimaryMarketplace.sol SHA3: 628e9bb16871d32c94ddfdbee26379b141b93636a34d9da72f8c074c File: contracts/ERC1155/interface/IERC1155.sol SHA3: 429819fa05cf0924f64d88384f597abdd54bb76ff2aff9b419c3cc0b File: contracts/ERC1155/interface/IFactory.sol SHA3: 19fdceb2e5fa75d204a873eb83455b44b287b5b5f9d8a498f07b8901 File: contracts/ERC1155/interface/ICollection.sol SHA3: 7e6ffeb8c3eba151286111de8a65870167eae149adb0b0e206307686