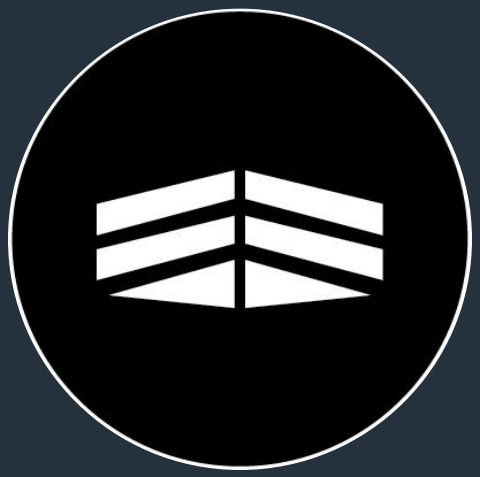


SMART CONTRACT AUDIT

ZOKYO.

June 29th 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

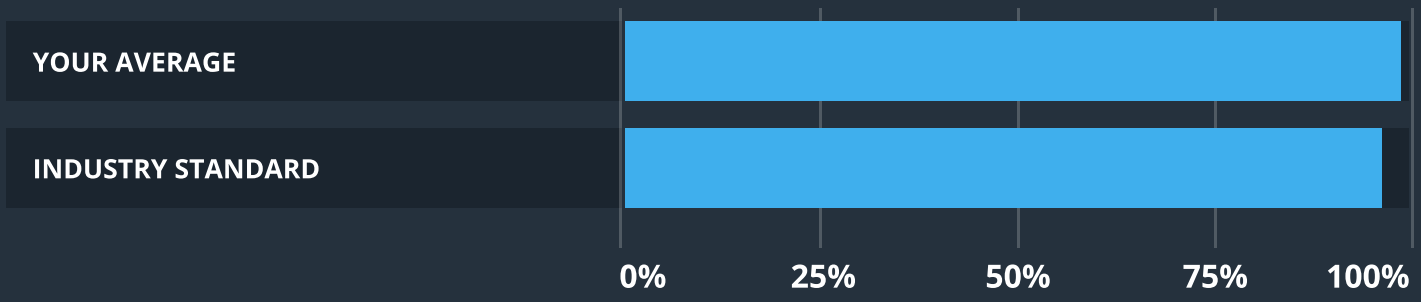
TECHNICAL SUMMARY

This document outlines the overall security of the Metropolis smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Metropolis smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Metropolis team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Metropolis repository.
https://github.com/Pips-Isalnd-Metropolis/ThePassport

Initial commit: aaf7ea4871412754298e9611540791a62576af0d
Last audited commit: a787273ecca0424db01505f3960a865d9ddf8443

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- AccessTokenContract.sol
- ImageData.sol
- PaymentSplit.sol
- SoftClayContract.sol
- TokenURI.sol
- VRFLottery.sol
- WLContract.sol
- WinChancesContract.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Metropolis smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

During the verification of the contract's set, auditor's team has found:
1) a serious vulnerability in the operation of the lottery, and separately in the formation of a random number
2) errors associated with the overpayment of the user as well as the distribution of funds received
3) errors with the distribution of access to methods
4) various errors breaking business logic under certain conditions
5) other errors and vulnerabilities, elements with the possibility of gas optimization.

The team found has added recommendations for possible solutions. Besides, most of the errors found were related to the unclear behavior of the business logic, or the failure of some functions. Though, Metropolis team has provided verifications or fixes for most of security related issues.

The Metropolis protocol is based on the ERC721 token (MetropolisWorldPassport contract), which is a user passport for the Metropolis universe. This passport is a repository of information about the user, which stores name, avatar, animation, rank, amount of soft clay, a list of buildings that the user owns, as well as the probability of winning the lottery.

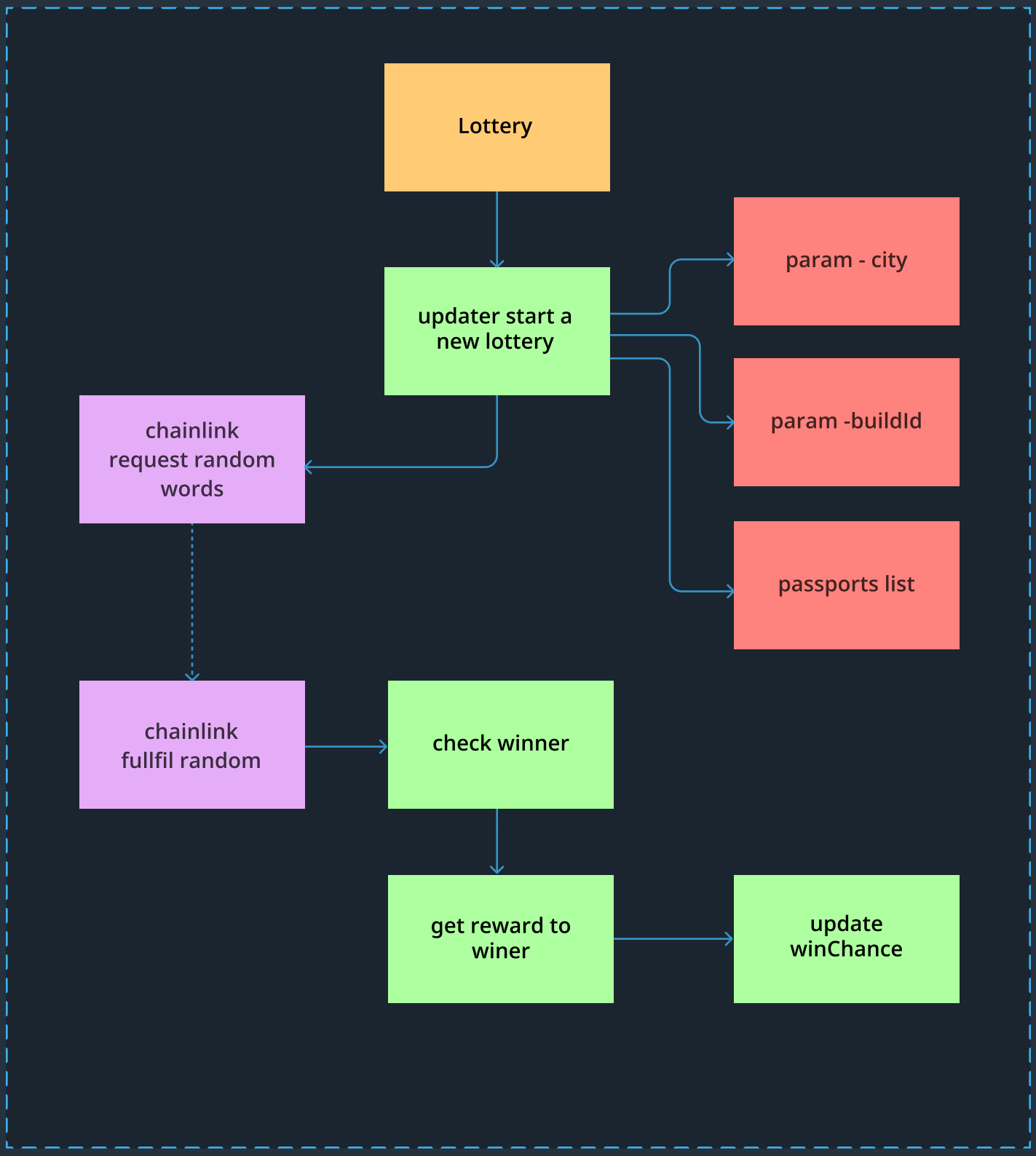
Upon receipt of a passport, the user will receive the opportunity to participate in lotteries (PropertyDraw contract) where the prize contains buildings in different cities of Metropolis (CityWL contract). The user has the opportunity to increase the probability of winning the next lottery (WinChances contract).

Soft clay is the internal currency of the Metropolis universe (SoftClay contract). It is charged to the user outside the blockchain, with the help of admins. With this currency, users can increase their rank.

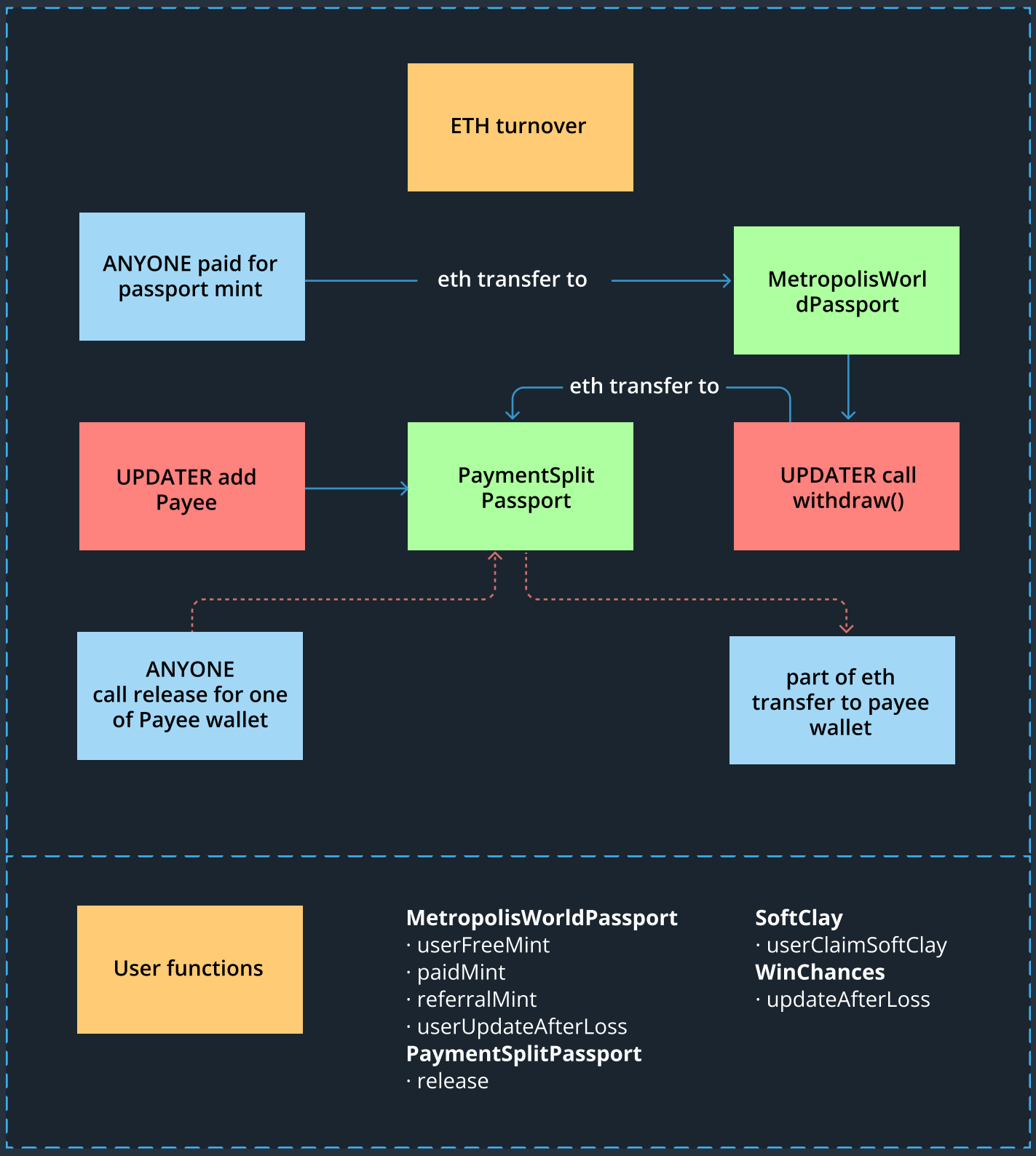
The cash flow of the project consists of the passport purchase by users (with ETH), and the subsequent transfer of these funds to a separate contract for the distribution (PaymentSplitPassport contract), where the total amount is distributed in accordance with the specified by admin rules.

PROTOCOL OVERVIEW
Mint Passport

Lottery


Lottery
updater start a new lottery
param - city
param -buildId
passports list
chainlink request random words
chainlink fullfil random
check winner
get reward to winer
update winChance

ETH Turnover



User functions

MetropolisWorldPassport
· userFreeMint
· paidMint
· referralMint
· userUpdateAfterLoss
PaymentSplitPassport
· release

SoftClay
· userClaimSoftClay
WinChances
· updateAfterLoss

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

CRITICAL | RESOLVED

Incorrect random source usage in the lottery

Getting a random value from the Chainlink is used incorrectly . The requestRandomWords() request receives a requestId and then must wait for a response transaction from the coordinator.
Only when the coordinator calls the fulfillRandomWords method, the
contract will receive a valid random value with which to determine the winner.

File:
VRFLottery.sol 112-120

```
    uint winner = passportIds[s_randomRange];
    //add the wl to the passport
    WLCont.addWLSpot(winner, city, buildingName, buildingId);
    //decrease win chances for winner
    WinCont.updateAfterWin(winner, city, buildingId);
    //loosers claim thier own increases
    //add to mapping
    _drawsDone[city][buildingId] = true;
    _winners[city][buildingId] = winner;
```

Recommendation:
This part of the logic should be separated and should be called in or after the fulfillRandomWords method

Post-audit:
The alternative proposed by the Metropolis team had a serious vulnerability in random number generation which was based on block.timestamp, block.difficulty, msg.sender values. So the info will become known to the miner at the time of the transaction mint and can be manipulated. So, Metropolis team has moved the lottery logic off chain.

From client:
We are likely going to take this off chain so I have added a function which allows us to manually add a winner, along with a recording of the draw that selected the winner.

CRITICAL | RESOLVED

Payment is calculated wrong in case a new payee is added.

PaymentSplit.sol
In case, a new payee is added, pending payment for other users will be calculated wrong in case they have released tokens before.
Example:
User has a share, equal to 10. Total shares is 100.
User1 calls release(user1Address).
totalReceived = 100 ETH.
payment = 100 * 10 / 100 - 0.
_released(user1Address) = 10
_totalReleased = 10.

After that, another user is added with a share, equal to 10. total shares is 110.
User1 calls release again.
totalReceived = 105 ETH. (Additional ETH was transferred to contract).
payment = 105 * 10 / 110 - 10 = 9.54 - 10. An underflow is occured, preventing user from claiming. Future payment might also be corrupted.

Recommendation:
Recalculate shares for users, so that no underflow occur and payments are paid as intended.

From client:
Added a control function so that the new payees can only be added before release is called. Release is blocked until we stop allowing new payees

CRITICAL | RESOLVED

Claimable soft clay amount can be claimed unlimited times.

SoftClayContract.sol: function userClaimSoftClay().
After a user has claimed soft clay, the value from mapping _claimableSoftClay is not set to zero, which means that user can claim the same soft clay multiple times.

Recommendation:
Set the value for mapping _claimableSoftClay[passportId] to zero after clay is claimed.

From client:
Fixed, now claimable amount is adjusted to zero once claimed.

HIGH | RESOLVED

Unable to access animations

AccessTokenContract.sol
ImageData.sol
Users will not be able to access animations at all.
First, Navigator and Pioneer level animations have no way to set them anywhere (neither in the constructor nor in the setter)._navAnnimation and _pionAnnimation never initialized, but, used in getAnnimationForElement
Secondly, when updating the passport rank to the level of the legend, the animation is not set.

Recommendation:
Initialize the animations of the Pioner and Navigation levels, or remove them by making a stub in the getter. Change the updateRank method by adding an animation setting to it, or make a separate setter, depending on the needs of the business logic.

From client:
This is now added: so the animation is added only once and will not change when the passport level changes. A function is added to the Image Contract to set the animation.

HIGH | VERIFIED

Users can get more passports per account than set in the limit.

In the contract, we have the _maxAllowedPerWallet parameter, which is responsible for the limit of passports per wallet. Verification is carried out in

File:
AccessTokenContract.sol 181: function _internalMint
require(balanceOf(toWallet) < _maxAllowedPerWallet, "address already owns max allowed");
The user can bypass this limitation by following a simple scenario:
The user mints the token. The user transfers the token to another address. The user mints the token again. This is especially important if the user was given the opportunity to mint for free.

Recommendation:
Since the ability to transfer/sell a passport to another user should be available, it may be necessary to either keep a list of users who are issued a passport, or limit the transfer by making the sale a separate controlled method.

From client:
Metropolis team understands this risk and it is allowable under our model. So users are allowed to mint, transfer and mint again. Also contracts allow users minting and buying on the secondary market. Given the current market the team will likely increase this limit anyway. Broadly this is an artificial constraint to encourage urgency.

Post-audit:
There is a risk that a user (especially a user with a free mint) may overmint the entire passport limit.

From client:
Additional checks added

HIGH | RESOLVED

No money back.

AccessTokenContract.sol, paidMint() and referralMint()
There is no handling in case the user pays more than required. Thus extra funds are not returned to the user.

Recommendation:
Add the processing of this case, returning the balance of the necessary funds to the user. Be careful and consider the reentrancy vulnerability.

Post-audit:
1. Add processing to the Metropolis World Passport.referralMint method
2. Move storage changes to the end of the function.
3. Use function safeTransferETH or Address.sendValue() instead of transfer

HIGH | RESOLVED

Eternal free mint.

AccessTokenContract.sol
There is no function to prevent the user from free minting. If the user is allowed a free mint, this is an endless, free opportunity to mint passports with increased winChance. Especially given the real absence of a limit on the limit, as well as bulkMint - this is a potential opportunity to bypass the entire business logic for payment very easily.

Recommendation:
Consider proper management of enabling users to mint for free. Add a fully functional restriction (at least for a free mint), or at least add a function to take away the user's ability to freely mint.

From client:
Eternal free mint now removed with the introduction of a free mint count check.

HIGH | VERIFIED

Not consolidated data

SoftClay.sol, setPioneerLevel(), setLegendLevel()
Consider carefully the situation with changing the levels of required softClay.
For example, if quantity 100 is needed for the Pioneer level first, and the user has 105 softClay units -> the Pioneer rank is assigned. Then you change the limit, and set the required level bar to 200. Verify if the user should stay at the Pioneer level in this case. Issue is set as High-risk, since there are no methods to update all fields as required.

Recommendation:
Verify the functionality of changing levels of required softClay

From client:
The Metropolis team has confiremed that it is allowable for the update of passport rank to occur only when the user next redeems or adds to their softClay balance.

HIGH | RESOLVED

Referral minter can pass his future token id to receive referral points. AccessTokenContract.sol: function referralMint().

Message sender can pre compute a newItemId, which will be minted to him and pass it as referrerTokenId, thus receive referral points for himself.
Also, having the opportunity to have several passports - the user can specify his referrerTokenId from his other passport.

Recommendation:
Verify that referrerTokenId exists and is not equal to newItemId.

MEDIUM | RESOLVED

The ability to increase winChance without participating in the lottery.

WinChancesContract.sol, updateAfterLoss()
If the user was not a lottery participant (for example, he joined later), he can still raise his winChance in the same way as if he lost in each of them.

Recommendation:
Verify the functionality of late users joined to the lottery.

From client:
Added contingency functions for this. So the team can manually stop the increasemet when we feel enough time has passed.

Post-audit:
Note that the protocol runs the lottery at the city level - not the buildings. That is, if the protocol blocks the lottery claim, and holds some kind of draw after a long time again in this city, this new lottery will open access to winChance increasement for new users by the amount equal to the number of buildings already sold in this city.

MEDIUM | RESOLVED

Updater without an admin role cannot add roles.

AccessTokenContract.sol, WinChances.sol, WLContract.sol: function addTheRolesForContractCalls().
Function calls grantRoles() function from AcessControl, which checks that the message sender is the default admin.
Thus, in case Updater doesn't have a default admin role, he won't be able to call addTheRolesForContractCalls().
If you assign UPDATER_ROLE to anyone other than the creator of the contract, that user will not be able to call these methods.

Recommendation:
Call internal function _grantRole() to avoid checking the default admin role.

MEDIUM | AKNOWLEDGED

Not profitable standard mint

AccessTokenContract.sol
There are two methods - paidMint and referralMint. Both methods are public. But in the referralMint method, the user sets himself more winChance (2 instead of 1), while you can specify any passportId, since this information is public.

Recommendation:
Depending on the needs of your business logic, you may want to make winChance the same in both cases. Or put a restriction on the use of referralMint, for example, by pre-mapping sponsor/referral. This approach will require more spending of users' money - but it will keep the business logic. Perhaps there are more options. Thus the verification is required.

From client:
The team is acknowledged of this and the concerns. From the business perspective Metropolis team allows the use case where a user chooses to use the referral mint instead of paid mint. The team can protect this a little on the front end and allows those who bypass the frontend to do this.

MEDIUM | RESOLVED

Not equal softClay amount

AccessTokenContract.sol
The updateRank method does not take into account cases where softClay is equal to any of the levels.

Recommendation:
Add variant handling when softClay is equal to levels.

MEDIUM | RESOLVED

Not working method bulkMint.

AccessTokenContract.sol function bulkMint()
The bulkMint method sequentially calls the _internalMint method, which in turn checks the user's passport balance limit. That is, calling BulkMint with the number of -5, but with a limit of 1 - the transaction will be terminated on the second call to _internalMint.

Recommendation:
Remove and redo the constraint (balanceOf(toWallet) < _maxAllowedPerWallet inside the _internalMint method

From client:
Added a check within bulk mint to ensure the number of mints will not break flow.

Post-audit:
function bulkMint
If you pass numberOfMints greater than mintLimit or maxAllowedPerWall then these checks will overflow.

```
    require(balanceOf(msg.sender) < _maxAllowedPerWallet - numberOfMints, "address already owns max allowed");
    require(_tokenIds.current() < _mintLimit-numberOfMints, "Will take you over max supply");
```

Thanks to the built-in overflow check in solidity ^8.0, it will throw an error, but since these are user-defined functions - it is recommended to add a check for (numberOfMints < maxAllowedPerWall), with the full text of the error so that users understand the problem.

So, the check for (_tokenIds.current() < _mintLimit-numberOfMints) can be replaced with (_tokenIds.current() + numberOfMints - 1 <= mintLimit)

LOW | VERIFIED

Soft Clay limit

There is a comment in the code indicating that softClay has a limit of 4 billion, but there are no checks for this. uint32 is a little bit more than 4 billion.

File:
AccessTokenContract.sol 72

```
    uint32 softClay; // max is 4 billion
```

Recommendation:
If this violates the business logic, add a limit check to the MetropolisWorldPassport.increaseSoftClay method

From client:
Metropolis team don't have a business need to set the limit at 4 billion as it is expected to release far below this amount, the note was to remind of the rough upper limit of the variable type.

LOW | RESOLVED

Inaccurate access.

The user can increase their winChance from both the MetropolisWorldPassport contract and the WinChances contract. In the case of the MetropolisWorldPassport contract, there is a check for possession of a passport - and in the other case, no.

Recommendation:
Make access in one place, and mandatory verification.

LOW | RESOLVED

Never used storage

PropertyDraw._start
CityWL._avatars
MetropolisWorldPassport._wlIds
Unused storage removal will decrease gas usage during the deployment and will increase the code quality.

Recommendation:
remove unused storage.

LOW | UNRESOLVED

Pragma version

Use the latest (if possible) version of pragma. Avoid floating pragmas for non-library contracts and use the fixed version of the latest stable release. As for now the latest stable version is 0.8.15. Usage of the floating point version increases risk to get the unstable release (as it was for several versions in 0.8.x line).

Recommendation:
use explicitly Solidity 0.8.15

INFORMATIONAL | AKNOWLEDGED

Centralized solution and great influence of manual control

Most of the functionality depends on the manual management of UPDATE_ROLE. Because of this, there is a high probability of making an error outside the contract part, which will get into the blockchain, and if it does not destroy the system, then at least it will lead to data inconsistency. Thus, the information about the role system should be present in the report.

Recommendation:
Add the description of the roles system to the documentation, make it clearly visible for users in the documentation about the admin functions, consider usage of the DAO or decentralized keepers.

From client:
A series of user facing docs is in the works to cover this.

INFORMATIONAL | RESOLVED

Extra memory usage

AccessTokenContract.sol, _internalMint()
You have:

```
string memory elm = elements[0];
uint x = newItemId % 6;
elm = elements[x];
```

Better:

```
string memory elm = elements[newItemId % 6];
```

Also consider usage of the opimized code:
avatarWl: newItemId % 2 == 1 ? _evenAvatar : _oddAvatar,

Recommendation:
consider memory usage optimization

INFORMATIONAL | RESOLVED

Unoptimized use of events.

Check all events throughout contracts. Somewhere you have events like this:
event AvatarWLSpotAdded(string update, string avatar, uint256 tokenId);
Field “update” contains information that duplicates the event name or could be placed in the name of the event. Thus it creates extra gas spending for the string construction and extra-parameter passing. Though the event is already a self-describing object.

Recommendation:
Consider revising the events system.

INFORMATIONAL | RESOLVED

Constants usage:

PropertyDraw._start (contracts/VRFLottery.sol#59)
PropertyDraw.callbackGasLimit (contracts/VRFLottery.sol#46)
PropertyDraw.keyHash (contracts/VRFLottery.sol#38)
PropertyDraw.numWords (contracts/VRFLottery.sol#53)
PropertyDraw.requestConfirmations (contracts/VRFLottery.sol#49)
PropertyDraw.vrfCoordinator (contracts/VRFLottery.sol#33)
Consider usage of constants, since these values are set only once

Recommendation:
Consider constants usage

Post-audit:
Contract PropertyDraw is no longer used.

External vs public

Use public modificator only if function can be called inside and outside contract. if it's used only from outside - use external, it's cheaper from the gas usage perspective.

- MetropolisWorldPassport.tokenURI(uint256) (contracts/AccessTokenContract.sol#461-471)
- MetropolisWorldPassport.contractURI() (contracts/AccessTokenContract.sol#139-141)
- MetropolisWorldPassport.addTheRolesForContractCalls(address,address,address,address) (contracts/AccessTokenContract.sol#167-172)
- MetropolisWorldPassport.freeMint(address) (contracts/AccessTokenContract.sol#214-216)
- MetropolisWorldPassport.userFreeMint() (contracts/AccessTokenContract.sol#221-225)
- MetropolisWorldPassport.paidMint() (contracts/AccessTokenContract.sol#230-240)
- MetropolisWorldPassport.bulkMint(uint16,address) (contracts/AccessTokenContract.sol#247-257)
- MetropolisWorldPassport.referralMint(uint256) (contracts/AccessTokenContract.sol#263-273)
- MetropolisWorldPassport.manualAddAvatarWL(uint256,string) (contracts/AccessTokenContract.sol#316-323)
- MetropolisWorldPassport.setAvatarWLNames(string,string) (contracts/AccessTokenContract.sol#329-335)
- MetropolisWorldPassport.userUpdateAfterLoss(uint256,string,uint32) (contracts/AccessTokenContract.sol#338-342)
- MetropolisWorldPassport.getWinChances(uint256) (contracts/AccessTokenContract.sol#353-356)
- MetropolisWorldPassport.setFreeMinters(address[]) (contracts/AccessTokenContract.sol#399-405)
- MetropolisWorldPassport.setPrice(uint256) (contracts/AccessTokenContract.sol#407-410)
- MetropolisWorldPassport.setMaxAllowed(uint16) (contracts/AccessTokenContract.sol#412-415)
- MetropolisWorldPassport.getMaxAllowed() (contracts/AccessTokenContract.sol#417-420)
- MetropolisWorldPassport.getDefaultAccessTokens() (contracts/AccessTokenContract.sol#422-429)
- MetropolisWorldPassport.getCurrentTokenId() (contracts/AccessTokenContract.sol#446-452)
- MetropolisWorldPassport.setMintLimit(uint32) (contracts/AccessTokenContract.sol#454-457)

- MetropolisWorldPassport.setContractURI(string,string,string,string,string) (contracts/AccessTokenContract.sol#482-508)
- MetropolisWorldPassport.withdraw() (contracts/AccessTokenContract.sol#511-514)
- ImageDataContract.setNavImages(string[],string[]) (contracts/ImageData.sol#53-64)
- ImageDataContract.setPioneerImages(string[],string[]) (contracts/ImageData.sol#66-77)
- ImageDataContract.setLegendImages(string[],string[]) (contracts/ImageData.sol#78-89)
- ImageDataContract.setAnnimations(string[]) (contracts/ImageData.sol#90-100)
- PaymentSplitPassport.getBalance() (contracts/PaymentSplit.sol#67-69)
- PaymentSplitPassport.totalShares() (contracts/PaymentSplit.sol#74-76)
- PaymentSplitPassport.shares(address) (contracts/PaymentSplit.sol#96-98)
- PaymentSplitPassport.payee(uint256) (contracts/PaymentSplit.sol#118-120)
- PaymentSplitPassport.releaseAll() (contracts/PaymentSplit.sol#145-150)
- PaymentSplitPassport.release(IERC20,address) (contracts/PaymentSplit.sol#157-170)
- PaymentSplitPassport.addPayee(address,uint256) (contracts/PaymentSplit.sol#189-191)
- SoftClay.awardSoftClay(uint256,uint32) (contracts/SoftClayContract.sol#37-41)
- SoftClay.userClaimSoftClay(uint256) (contracts/SoftClayContract.sol#43-49)
- SoftClay.addClaimableSoftClay(uint256[],uint32[]) (contracts/SoftClayContract.sol#51-55)
- SoftClay.getClaimableClay(uint256) (contracts/SoftClayContract.sol#57-59)
- SoftClay.setPioneerLevel(uint32) (contracts/SoftClayContract.sol#71-74)
- SoftClay.setLegendLevel(uint32) (contracts/SoftClayContract.sol#76-79)
- SoftClay.getPioneerLevel() (contracts/SoftClayContract.sol#81-84)
- SoftClay.getLegendLevel() (contracts/SoftClayContract.sol#86-89)
- PropertyDraw.addTheRolesForContractCalls(address,address) (contracts/VRFLottery.sol#36-39)
- PropertyDraw.setPartnerContracts(address,address) (contracts/VRFLottery.sol#41-46)
- CityWL.addTheRolesForContractCalls(address,address) (contracts/WLContract.sol#58-61)
- CityWL.removeCityWlSpot(uint256,string,uint32) (contracts/WLContract.sol#78-83)
- WinChances.addTheRolesForContractCalls(address,address) (contracts/WinChancesContract.sol#53-56)
- WinChances.setLossIncrease(uint16) (contracts/WinChancesContract.sol#83-87)
- WinChances.setReferralIncrement(uint16) (contracts/WinChancesContract.sol#89-93)
- WinChances.setWinDecrease(uint16) (contracts/WinChancesContract.sol#95-99)
- WinChances.closeCityForWinLossRedemption(string) (contracts/WinChancesContract.sol#113-115)

Recommendation:
Change the modifier in the specified methods to external if these methods are not going to be called within this contract.

Post-audit:
Left unchanged:

- MetropolisWorldPassport.tokenURI(uint256) (contracts/AccessTokenContract.sol#447-457)
- MetropolisWorldPassport.contractURI() (contracts/AccessTokenContract.sol#123-125)
- MetropolisWorldPassport.setMintLimit(uint32) (contracts/AccessTokenContract.sol#440-443)
- PaymentSplitPassport.payee(uint256) (contracts/PaymentSplit.sol#119-121)
- PaymentSplitPassport.releaseAll() (contracts/PaymentSplit.sol#147-153)
- PaymentSplitPassport.addPayee(address,uint256) (contracts/PaymentSplit.sol#182-185)
- PropertyDraw.addTheRolesForContractCalls(address,address) (contracts/VRFLottery.sol#37-42)
- PropertyDraw.setPartnerContracts(address,address)

INFORMATIONAL | RESOLVED

Missing zero-address checks.

There are no checks for a zero address when initializing contracts.
MetropolisWorldPassport - constructor, setWLContractAddress
SoftClay - constructor
PropertyDraw - setPartnerContracts
WinChances - constructor
CityWL - constructor

Recommendation:
You should add checks for the zero address when initializing the contracts and payment addresses.

INFORMATIONAL | RESOLVED

Calldata vs memory

Use calldata for all input parameters instead of memory. Use memory only if you change this parameter inside the function.
ImageDataContract

- setNavImages(string[] memory image, string[] memory cdnImage)
- setPioneerImages(string[] memory ipfs, string[] memory cdn)
- setLegendImages(string[] memory ipfs, string[] memory cdn)
- setAnnimations(string[] memory animations)

Recommendation:
Change the modifier in the specified methods to calldata.

INFORMATIONAL | RESOLVED

Re-inheritance

Don`t need to inherit MetropolisWorldPassport from ERC721, because ERC721Enumerable already inherits from ERC721.

Recommendation:
Remove inheritance from ERC721 in MetropolisWorldPassport contract.

String alternative

Consider using an enum instead of an array of strings for elements and avatar. Using string is always more vulnerable to management errors and takes up more memory.
Potential replacement variables:
File: AccessTokenContract.so

- 64: string private _oddAvatar = "nomad"; 65: string private _evenAvatar = "citizen"; could be: enum Avatars{ Nomad, Citizen }
- 84: string[] elements = ["Fire", "Water", "Air", "Space", "Pixel", "Earth"]; could be: enum Elements{ Fire, Water, Air, Space, Pixel, Earth }
- ranks in MetropolisWorldPassport could be: enum Ranks{ Navigator, Pioneer, Legend, }

Recommendation:
Replace parameters of type string with elements of structures of type enum.

From client:
Avatars will change over time, so needs to remain updatable.

INFORMATIONAL | UNRESOLVED

Unnecessary private fields, duplicated getters

You have private fields in your contacts for which there are public getters. If you do not plan to inherit from these contracts, then you can probably remove unnecessary getters and make the variables public. Solidity will automatically create a public getter for such public variable. There are also public variables for which you also wrote a direct getter. Basically, it's code duplication. Thus following the recommendation will reduce the contract size and increase readability.
Unnecessary private fields:
MetropolisWorldPassport._contractURI
MetropolisWorldPassport._maxAllowedPerWallet
PaymentSplitPassport._totalShares
PaymentSplitPassport._totalReleased
PaymentSplitPassport._erc20TotalReleased
PaymentSplitPassport._shares
PaymentSplitPassport._released
PaymentSplitPassport._erc20Released
PaymentSplitPassport._payees
SoftClay._pioneerLevel
SoftClay._legendLevel
PropertyDraw._drawsDone
WinChances._citiesDropped
Duplicated getters:
WinChances.getReferalIncrease

Recommendation:
Reconsider whether you need a private modifier in all these fields. Put the public modifier where possible, and remove the manual getter.
Consider removing manual getters for public variables.

INFORMATIONAL | UNRESOLVED

Unreleased ERC20

The PaymentSplitPassport contract contains methods for ERC20 tokens, but the general scheme for managing funds in other contracts (MetropolisWorldPassport) does not have the ability to buy for ERC20, as well as send ERC20 to the PaymentSplitPassport contract. If this is not your idea and the system will not use the ERC20 token in the future (without deploying a new PaymentSplitPassport contract), then you should remove unnecessary functions and variables.
File: PaymentSplitPassport.sol
_erc20TotalReleased
_erc20Released
function release(IERC20 token, address account) public virtual
function totalReleased(IERC20 token) public view returns (uint256)
function released(IERC20 token, address account) public view returns (uint256)

Recommendation:
Consider removing the specified methods and variables or verify the correctness of ERC20 tokens usage.

Check that the length of array parameters matches.

SoftClayContract.sol: function addClaimableSoftClay().
The length of passportIds and amounts should be verified to match each other. Such validations are helpful for ease of error navigation.
ImageData.sol: function setNavImages()
Length of image and cdnImage should be verified to match each other
ImageData.sol: function setPioneerImages(), function setLegendImages()
Length of ipfs and cdn should be verified to match each other.
Raising these events will raise an error if the parameters are an empty array.
ImageData.sol:
emit NavImagesUpdated(image[0], cdnImage[0]);
emit PionImagesUpdated(ipfs[0], cdn[0]);
emit LegImagesUpdated(ipfs[0], cdn[0]);
emit AnimationsUpdated(animations[0]);

Recommendation:
Validate that length of specified arrays matches. Add a check for parameter equality to empty arrays.

Code optimization

File: ImageData.sol

```
   function getAnnimationForElement(string calldata element, uint16 level)external view
returns(string memory){
       if (level==1){
          return _navAnnimation[element];
       }else if (level==2){
          return _navAnnimation[element];
       }else{
          return _navAnnimation[element];
       }
   }
```

Better:

```
   function getAnnimationForElement(string calldata element, uint16 level)external view
returns(string memory){
       return _navAnnimation[element];
   }
```

Recommendation:
Change the code to be more economical and shorter.

	AccessTokenContract.sol	ImageData.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	PaymentSplit.sol	SoftClayContract.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	TokenURI.sol	VRFLottery.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	WLContract.sol	WinChancesContract.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Metropolis team

As part of our work assisting Metropolis team in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the Metropolis team.

It needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the Metropolis team. All of them were also carefully checked by the auditors' team.

Access Token contract

Correct setup

- ✓ should be named 'Met World Access Token'
- ✓ should have token id of 1

Minting Passports and checks

- ✓ should Mint a free NFT (Level 1) (296ms)
- ✓ should Mint a paid NFT (133ms)
- ✓ Should Mint a passport with referall that increases win chances by 2 (145ms)
- ✓ should mint a passport for free (user) mint (111ms)

Adding and removing WL spots on the the Passport

- ✓ should run property draw and add WL to winner (246ms)
- ✓ should add city WL spot to pasport (110ms)
- ✓ should add a character WL spot to the passport (89ms)
- ✓ should remove city wl spot from passport (116ms)
- ✓ should remove avatar wl spot from passport (96ms)
- ✓ should return the avatar wl spot for a passport (71ms)

Increasing and decreasing win chances

- ✓ should change win chance after lose
- ✓ should increase the win chance of a passport (84ms)
- ✓ should set the increse in win chance received by a loss
- ✓ should set the increment amount referrals increase win chances
- ✓ should set the amount win chances are decreased by a win
- ✓ should return the number a referral will increment win chances
- ✓ should return the number by which win chances will decrease after a win
- ✓ should return the amount a loss will increase win chances
- ✓ should get the win chances of a specific passport (68ms)

Adding and removing Soft Clay

- ✓ should set claimable clay for 3 passsports (233ms)
- ✓ should allow user to claim thier softclay (140ms)

```
    ✓ should award soft clay to a passport (110ms)
    ✓ should redeem soft clay (172ms)
    ✓ should set the level to achive pioneer
    ✓ should set the level to achieve legend
    ✓ Should update the image for the legend rank (58ms)
    ✓ Should update the images for the pioneer rank (51ms)
    ✓ should return the soft clay needed to make pioneer
    ✓ should return the soft clay needed to reach legend
  Admin and helper functions
    ✓ should update the price
    ✓ should set the max allowed per wallet
    ✓ should check which NFT's a wallet has (82ms)
    ✓ should return the token id of the next mint
    ✓ should returnt the current price of minting
    ✓ should return the current minting limit
    ✓ should set the limit for number of mints
    ✓ should update the images for new mints (82ms)
    ✓ should return the tokenURI of specified token (173ms)
    ✓ should set the contractURI for the collection (70ms)
  Access roles worrking as expected
    ✓ should fail when caller is not UPDATER_ROLE (122ms)
    ✓ should add new UPDATER_ROLE
    ✓ should add new BALANCE_ROLE
    ✓ should revoke UPDATER_ROLE
  ETH can be moved out
    ✓ should withdraw the balance from the contract (98ms)
    ✓ should add a new wallet to receive share
    ✓ should share proceeds of sale between share holders (164ms)
  Payment Splits contract
    ✓ should release payments to all listed
    ✓ should add a new payee to the list
  Errors and failed mints
    ✓ should indicate not enough eth
    ✓ should fail if tries to mint more then max per wallet(1) (91ms)
    ✓ should fail if to many total minted (105ms)
    ✓ should fail to remove avatar WL if owner not owns passport (79ms)
  54 passing (1m)
```

FILE	% STMTS	% BRANCH	% FUNCS
AccessTokenContract.sol	78.07	48.21	86.49
ImageData.sol	83.33	50	87.5
PaymentSplit.sol	94.74	50	84.62
SoftClayContract.sol	100	50	100
TokenURI.sol	100	100	100
VRFLottery.sol	91.67	50	85.71
WLContract.sol	79.31	40	83.33
WinChancesContract.sol	74.19	25	63.64
All files	**83.55**	**46**	**85.11**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting Metropolis in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Metropolis contract requirements for details about issuance amounts and how the system handles these.

Audit tests

AccessToken Contract

- ✓ should be reverted deploy if image contract equal zero address (49ms)
- ✓ should be reverted deploy if paymentSplit contract equal zero address (45ms)
- ✓ should fail if too many totals minted in freeMint method (60ms)
- ✓ should be reverted if user is not on the free mint list
- ✓ should be set avatar WL names (154ms)
- ✓ should return the correct avatar for the user (153ms)
- ✓ should supportsInterface return true
- ✓ should supportsInterface return false
- ✓ should be updated after loss (270ms)
- ✓ should be updated user passport after loss (272ms)
- ✓ should be reverted if the user doesn't have a passport (159ms)
- ✓ hould mint more than one passport (152ms)
- ✓ should be reverted if not draw city (157ms)
- ✓ should be reverted if the winner call updateAfterLose (169ms)
- ✓ should be reverted if the user call updateAfterLose more than once (181ms)
- ✓ should be reverted if msg.value is not enough for referralMint (56ms)
- ✓ should be reverted if msg.value is not enough for paidMint
- ✓ should be reverted if msg.value is not enough for bulkMint
- ✓ should be a mint passport with bulkMint (61ms)
- ✓ should be a mint more one passport with bulkMint (107ms)
- ✓ should be reverted updateRank if the user doesn't have a passport
- ✓ should return the excess ether bulkMint (64ms)
- ✓ should return the excess ether paidMint (58ms)
- ✓ should return the excess ether referralMint (126ms)
- ✓ should be reverted if the user wants to mint more than allowed (84ms)
- ✓ should be reverted if the user wants to mint more than mintLimit
- ✓ should set image contract

✓ should be reverted if image contract equal zero address
✓ should be reverted if winChanse contract equal zero address
✓ should be reverted if WL contract equal zero address
✓ should be reverted if TokenURI contract equal zero address

referral mint scenario

✓ should be reverted if the user sets his future passport id
✓ should be reverted if the user sets zero id

WinChances contract

✓ should reverted WinChanses contract deploy if passed zero address
✓ should be increased dropped cities
✓ should be reverted if the draw has not taken place (69ms)
✓ should be reverted if the city closed for claim chance (222ms)

SoftClay contract

✓ should be reverted SoftClay contract deploy if pass contract equal zero address
✓ should be reverted if the user doesn't have claimable soft clay (60ms)
✓ should be reverted if redeem more soft clay if the user has (83ms)
✓ should be returned SoftClay for a passport (60ms)
✓ should be reverted if passport ids array length does not equal amounts array (112ms)

soft clay scenario

✓ should not allow withdrawing soft clay many times (211ms)

WL contract

✓ should reverted WL contract deploy if passed zero address
✓ should be reverted add roles for contract calls if passed zero address
✓ should find wlSpot in the user passport (276ms)
✓ should be false if the user doesn't have WlSpot (68ms)
✓ should be reverted if the city wl slot does not exist (144ms)

Lottery Contract

✓ should be reverted add roles if passed zero address
✓ should be reverted add partners contract if passed zero address
✓ should be returned draw recording (175ms)
✓ should be reverted if city draws are done (207ms)
✓ should be verified winner (202ms)
✓ should be verified draw (208ms)

ImageData contract

✓ should be reverted contract deploy if images are not enough
✓ should be updated images uri for pioneer level (176ms)
✓ should be updated images uri for legend level (174ms)
✓ should be updated animations (47ms)
✓ Should be reverted if animations are not enough
✓ Should be reverted if pioneer images are not enough
✓ Should be reverted if navigator images are not enough
✓ Should be reverted if legend images are not enough

✓ Should be reverted if legen cdn images are not enough
✓ Should be reverted if pioneer cdn images are not enough
✓ Should be reverted if navigator cdn images are not enough
✓ should be reverted if navigator images array length equal zero
✓ should be reverted if pioneer images array length equal zero
✓ should be reverted if legend images array length equal zero
✓ should be updated images uri for navigator level (179ms)

PaymentSplit contract

✓ should be reverted deploy if payees array length is less than the shares array length
✓ should be reverted deploy if payees array length equal to zero
✓ should be reverted eth release if the account does not have shares
✓ should be reverted eth release if contract not close for new payees
✓ should be reverted eth releaseAll if contract not close for new payees
✓ should be reverted eth release if the contract does not have eth
✓ should be returned correct payees
✓ should return correct shares for payees
✓ should be reverted if payee to equal zero address
✓ should be reverted if shares equal zero
✓ should be reverted if the user has shares

underflow scenario

✓ should be paid ETH (238ms)

135 passing (2m)

FILE	% STMTS	% BRANCH	% FUNCS
AccessTokenContract.sol	100	96.43	100
ImageData.sol	100	100	100
PaymentSplit.sol	100	100	100
SoftClayContract.sol	100	100	100
TokenURI.sol	100	100	100
VRFLottery.sol	100	100	100
WLContract.sol	100	100	100
WinChancesContract.sol	100	93.75	100
All files	**100**	**98**	**100**

We are grateful to have been given the opportunity to work with the Metropolis team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Metropolis team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.